

Kommerzielle Entwicklung von Open-Source-Software: Idealismus, Pragmatismus oder Strategie?

Eine Fallstudie
über Entwickler des Linux-Kernels
in großen Firmen der Informationstechnologie

vorgelegt von lic. rer. pol.

Urs Lerch

aus Muttenz / Schweiz

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Wirtschaftswissenschaften
– Dr. rer. oec. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. J.-P. Seifert

Berichter: Prof. Dr. B. Lutterbeck (i.R.)

Berichter: Prof. Dr. H. Krallmann

Tag der wissenschaftlichen Aussprache: 23. April 2010

Berlin 2010

- D 83 -

Kommerzielle Entwicklung von Open-Source-Software: Idealismus, Pragmatismus oder Strategie?

Eine Fallstudie
über Entwickler des Linux-Kernels
in großen Firmen der Informationstechnologie

Urs Lerch

Berlin, 2010



Diese Arbeit steht unter einer
Creative Commons Namensnennung 3.0 Deutschland Lizenz.

„The answer is Linux, now what was the question again?“

(Vile und Atherton 2009, S. 10)

Zusammenfassung

Free/Libre and Open-Source-Software (FLOSS) im Allgemeinen sowie das Betriebssystem Linux im Speziellen erfreuen sich seit rund zehn Jahren zunehmender wirtschaftlicher Bedeutung und sind heute ein wesentlicher Baustein der professionellen Informations- und Kommunikationstechnologie. Entwickelt wird die Software mit reger Beteiligung einer Online-Community und zahlreichen Beiträgen von Firmen, Universitäten, Stiftungen sowie Freiwilligen. Die bislang geführte Diskussion zu FLOSS, beginnend im Jahre 1998, kann in drei zeitlich aufeinander folgende, jeweils ca. drei bis fünf Jahre dauernde Phasen gegliedert und mit den Überschriften „Idealismus“, „Pragmatismus“ und „Strategie“ umschrieben werden. Trotz der zunehmenden wirtschaftlichen Bedeutung von FLOSS auf der Nachfrage- wie auf der Angebotsseite wird in wissenschaftlichen Arbeiten weiterhin auf die sozialromantische und idealisierende Frühphase rekurriert. Die Unterstützung bei der FLOSS-Produktion durch große, etablierte Unternehmen der Informationstechnologie (IT) unter wirtschaftlichen Aspekten blieb bisher in der wissenschaftlichen Diskussion noch weitgehend unbeachtet. Diese Dissertation beabsichtigt, einen empirischen Beitrag zu dieser lückenhaften Datenlage zu leisten.

Anhand einer Fallstudie mit quantitativen und qualitativen methodischen Ansätzen wird die Situation von Linux-Kernel-Entwicklern in großen IT-Firmen untersucht. Die quantitative Analyse sämtlicher Logdateien des Linux-Kernels des Jahres 2007 hat ergeben, dass dieser zu rund drei Vierteln von Firmen weiterentwickelt wird. Mittels qualitativer, teilstandardisierter Interviews mit siebzehn Linux-Kernel-Entwicklern aus großen IT-Firmen kann eine Typologie von kommerziellen Open-Source-Entwicklern hergeleitet werden. Die Typen sind: der „Pragmatische Ingenieur“, der „Dialektische Informatiker“ und der „Sozialromantische Hacker“. Anhand dieser Typologie kann der Berufsalltag eines Open-Source-Entwicklers in einer großen IT-Firma vertiefend beschrieben werden.

Die empirischen Daten legen nahe, dass die Firmenbeteiligung bei Freier und Open-Source-Software weit höher ist als aus anderen Studien bekannt. Deshalb sollte das Open-Source-Entwicklungsmodell um die Firmenbeteiligung erweitert werden, insbesondere unter Berücksichtigung der Typologie der kommerziellen Open-Source-Entwickler. Zudem kann, basierend

auf den Daten, aufgezeigt werden, dass Freie und Open-Source-Software weitgehend mit bestehenden ökonomischen Theorien erklärbar ist, so dass das Image der Sozialromantik stark relativiert werden muss. Empfohlen wird deshalb, die Diskussion über FLOSS zukünftig weniger nach ideologisch orientierten Gesichtspunkten, sondern mit vorwiegend ökonomischen Argumenten zu führen. Dabei ist insbesondere in der ökonomischen Diskussion der freien Lizenz der Software mehr Beachtung zu schenken. Die Mitarbeit in einer Open-Source-Community sollte zudem aufgrund der Bedeutung von FLOSS in den Lehrplan der Informatikausbildung integriert werden.

Abstract

Free/Libre and Open Source Software (FLOSS) in general and the Linux operating system in particular have increased in economic significance during the past decade and are now an integral component of professional information and communication technology (IT). The software is continuously developed by an online community with numerous contributions from companies, universities, foundations and volunteers. The discussion on FLOSS started in 1998. Retrospectively, three consecutive phases, lasting three to five years each, are identified as “idealism”, “pragmatism” and “strategy”. Along increasing economic growth of FLOSS on demand and supply of software, scientific knowledge remains scarce, mainly due to the focus on social romantic and idealized positions in early stages. On the other hand, cooperation in FLOSS production by large, established IT companies, especially in economic terms, has been largely ignored in the scientific debate. This PhD project intends to make an empirical contribution to this gap.

The situation of Linux kernel developers in large IT companies is investigated in a case study with quantitative and qualitative methodological approaches. Quantitative analysis of all log files of the Linux kernel in the year 2007 shows, that commercial companies contribute 75 percent to the kernel development. Using qualitative, semi-structured interviews with seventeen Linux kernel developers from large IT companies, a typology of commercial FLOSS developers is identified. The three types are: “pragmatic engineer”, “dialectical computer scientist”, and “social romantic hacker”. Based on this typology, the working and personal environment of professional FLOSS developers in a large IT company is analyzed.

Empirical data from this study suggest that companies participate to FLOSS considerably more than identified in prior studies. Therefore, the open source development model should be expanded to business contributions. This process should include the identified typology of commercial FLOSS developers. In addition, FLOSS is largely explained by existing economic theories. Thus, the previous image of social romanticism needs to be reconsidered. And it is recommended that in discussing FLOSS, an economic perspective is integrated rather than reiterate ideologically oriented positions. On the other hand, the freedom granted by the license has to be more emphasized in the economic debate. Finally, regarding the importance of FLOSS, participation in an open source community should be included in the curriculum of computer science education.

Danksagung

Nach dem Abschluss meines Wirtschaftsstudiums an der Universität Basel kam ich in meiner mehrjährigen Projektstätigkeit im betriebsinternen Controlling sowie durch die Mitarbeit im Human-Ressource-Bereich zunehmend mit der Informatik in Berührung. Fasziniert von der Programmierung, beschloss ich vor mittlerweile fünfzehn Jahren, von der Anwendungsseite zur Entwicklung zu wechseln. Den Entschluss habe ich nie bereut, insbesondere weil mich die Hilfsbereitschaft und Kollegialität im Informatik-Umfeld positiv überrascht haben – Eigenschaften, die von außerhalb der IT eher als elitär und unprofessionell gewertet werden.

Ähnliches durfte ich an der TU Berlin am Lehrstuhl „Informatik und Gesellschaft“ erfahren. Beim Team um Bernd Lutterbeck möchte ich mich für die unkomplizierte und kollegiale Unterstützung ganz herzlich bedanken, wenn auch der Kontakt aufgrund meines Wohnortes in der Schweiz nicht so intensiv war, wie von mir gewünscht. Auch die zahlreichen Kontakte – meist per E-Mail –, die ich weltweit in der akademischen und in der Open-Source-Welt geknüpft habe, waren von großer Hilfsbereitschaft und Kollegialität geprägt. Nicht unterlassen möchte ich es, den fünf Firmen und insgesamt vierundzwanzig Personen, mit denen ich Interviews führen durfte, für ihr entgegengebrachtes Vertrauen und die großzügig zur Verfügung gestellte Zeit zu danken.

Ganz besonders danke ich Bernd Lutterbeck, dass er mich nach beinahe zwanzigjähriger akademischer Abstinenz – und deshalb weitgehend „entwissenschaftlicht“ – mit meinem Dissertationsvorhaben unterstützte. Er ermöglichte mir für die zweite Hälfte meines Berufslebens eine hervorragende Gelegenheit, die es „zu packen“ galt. Die Rahmenbedingungen waren wegen der geografischen Distanz zwar nicht besonders günstig. Trotzdem konnte ich auf elektronischem, telefonischem und natürlich auch persönlichem Weg die nötigen Fragen stets zeitnah diskutieren. Bernd, das entgegengebrachte Vertrauen habe ich außerordentlich geschätzt.

Meiner Frau Iren Bischofberger danke ich dafür, dass sie fast während der ganzen Promotionszeit für das Familieneinkommen aufkam und mir so einen großen Handlungsspielraum ermöglichte. Zudem waren ihre Kommentare insbesondere zur Methodik, aber auch bei der

kritischen Durchsicht der Kapitel sowie ihre permanente Unterstützung im Alltag eine unverzichtbare Hilfe. Sie erwirbt sich deshalb nach ihrem eigenen PhD noch einen PhT („push him through“). Unserer fünfjährigen Tochter Tina danke ich dafür, dass sie unser Leben täglich bereichert. Mit ihrer neugierigen, manchmal fordernden Art und ihrem Hinterfragen von für uns Erwachsene Selbstverständlichem hat sie dafür gesorgt, dass mir der Familienalltag eine hilfreiche Distanz zur akademischen Welt ermöglicht.

Inhaltsverzeichnis

1	Einführung in die Problemstellung.....	1
1.1	Ausgangslage.....	1
1.2	Theoretische Abgrenzung.....	3
1.3	Forschungsziel und erwarteter Beitrag.....	4
1.4	Exkurs: „Freie“ vs. „Offene“ Software.....	4
1.5	Aufbau der Studie.....	6
	Teil I: Theoretischer Hintergrund.....	7
2	Der Fall Linux.....	9
2.1	Der Kernel und die Distribution.....	9
2.2	Der Kernel in Zahlen.....	10
2.3	Entwicklungsmodell des Linux-Kernels.....	12
2.4	Herausforderungen innerhalb der Community.....	15
2.5	Zwischenfazit.....	17
3	Open-Source-Software – Stand der Diskussion.....	19
3.1	Vorgeschichte.....	19
3.2	Die erste Phase: Idealismus.....	22
3.2.1	Der Basar: Ein neues Entwicklungsmodell.....	23
3.2.2	Motivation in der Open-Source-Software-Entwicklung.....	29
3.2.2.1	Intrinsische und extrinsische Motivation.....	30
3.2.2.2	Führung über die Motivation.....	36
3.2.2.3	Opportunitätskosten.....	36
3.2.2.4	Rationale und evolutionäre Entscheidungstheorie.....	37
3.2.3	Zwischenfazit.....	39
3.3	Die zweite Phase: Pragmatismus.....	40
3.3.1	Motivation aus Firmenperspektive.....	41
3.3.2	Neue Geschäftsmodelle.....	42
3.3.2.1	Produkt-Geschäftsmodelle.....	43
3.3.2.2	Dienstleistungs-Geschäftsmodelle.....	45
3.3.3	Eine neue Form der Organisation.....	46
3.3.4	Zwischenfazit.....	47
3.4	Die dritte Phase: Strategie.....	48
3.4.1	„Openness“: Ein erweitertes Verständnis.....	49
3.4.2	Der Infrastrukturbegriff.....	51
3.4.3	Ein Reifeprozess-Modell.....	53
3.4.4	Mitarbeit in kommerziellen Firmen.....	57
3.4.5	Zwischenfazit.....	59
3.5	Konklusion zum Stand der Diskussion.....	60
3.6	Forschungsfragen.....	62

Teil II: Methodisches Konzept	63
4 Methodische Hinführung	65
4.1 Fallstudien-Design.....	65
4.2 Gütekriterien qualitativer Forschung.....	66
5 Quantitative Erhebung der Linux-Kernel-Logfiles	69
5.1 Ausgangslage.....	69
5.2 Sampling.....	71
5.3 Datenerhebung.....	72
5.4 Datenanalyse.....	73
6 Qualitative Experteninterviews von Linux-Kernel-Entwicklern in großen IKT-Firmen	75
6.1 Das Experteninterview.....	75
6.1.1 Wer ist ein Experte?.....	75
6.1.2 Rolle des Interviewenden.....	76
6.2 Sampling.....	77
6.3 Rekrutierungsstrategie.....	78
6.4 Leitfadenentwicklung.....	80
6.5 Datenerhebung.....	81
6.6 Datenanalyse.....	82
6.6.1 Transkription.....	83
6.6.2 Induktives Codieren.....	84
6.7 Typenbildung.....	85
6.8 Kontrolle der Datenanalyse.....	87
6.9 Grenzen des methodischen Vorgehens.....	87
Teil III: Ergebnispräsentation	89
7 Ergebnisse zu den Linux-Kernel-Logfiles	91
7.1 Umfang der Beiträge.....	91
7.2 Interessengruppen.....	92
7.3 Beteiligte Firmen.....	93
7.3.1 Top-Firmen.....	93
7.3.2 Firmen nach Regionen.....	94
7.3.3 Firmen nach Größe.....	96
7.3.4 Firmen nach Sektoren.....	97
7.3.5 Firmen nach Anzahl der Linux-Entwickler.....	97
7.4 Kategorisierung der Firmenbeiträge.....	99
7.5 Aufteilung nach Architekturen.....	102

8	Linux-Kernel-Entwickler in großen IKT-Firmen.....	107
8.1	Soziodemografischer Überblick.....	107
8.2	Die empirisch begründete Typenbildung.....	109
8.3	Der pragmatische Ingenieur.....	111
8.3.1	Charakterisierung des Prototyps: Franz (F6).....	111
8.3.1.1	Werdegang.....	111
8.3.1.2	Technologische Orientierung.....	112
8.3.1.3	Ein Job wie jeder andere.....	113
8.3.1.4	Wenig Präsenz in der Open-Source-Community.....	114
8.3.1.5	Firmeninteressen haben Priorität.....	115
8.3.1.6	Den Status quo erhalten.....	115
8.3.2	Idealtypische Charakterisierung.....	116
8.4	Der dialektische Informatiker.....	120
8.4.1	Charakterisierung des Prototyps: Gerhard (G7).....	120
8.4.1.1	Werdegang.....	120
8.4.1.2	Etwas bewegen können.....	121
8.4.1.3	Firma und Community zusammenbringen.....	121
8.4.2	Idealtypische Charakterisierung.....	123
8.5	Der sozialromantische Hacker.....	127
8.5.1	Charakterisierung des Prototyps: Robert (R18).....	127
8.5.1.1	Werdegang.....	127
8.5.1.2	Eine neue emotionale Heimat wird gesucht und gefunden.....	128
8.5.1.3	Zunehmende Entfremdung in der Arbeit.....	129
8.5.1.4	Ausweichen auf andere Schauplätze.....	130
8.5.2	Idealtypische Charakterisierung.....	131
Teil IV: Diskussion, Schlussfolgerungen & Ausblick.....		133
9	Diskussion.....	135
9.1	Firmenbeteiligung bei FLOSS.....	136
9.1.1	Ökonomische Interessen.....	136
9.1.2	Ökonomische vs. ideelle Argumentation.....	138
9.1.3	FLOSS und ökonomische Trends.....	143
9.1.4	Strukturelle Folgerungen.....	145
9.1.4.1	Digitaler Graben.....	145
9.1.4.2	Unternehmerischer Mittelstand.....	146
9.1.5	Erweiterung des Infrastrukturbegriffs.....	147
9.2	Ein erweitertes Basar-Modell.....	150
9.2.1	Motivation von kommerziellen FLOSS-Entwicklern.....	152
9.2.2	Kommerzielle Rollen in FLOSS-Projekten.....	153
9.2.2.1	Einordnung der Typologie in die Community-Struktur.....	153
9.2.2.2	Wechselwirkungen bei den Typen.....	157
9.2.2.3	Der Umgang der verschiedenen Entwicklertypen mit Konflikten.....	159
9.2.2.4	Transitionen der Typen.....	163
9.2.3	Vertraglich abgesicherte Selbstorganisation.....	166

10	Schlussfolgerungen.....	171
11	Ausblick.....	177
12	Literaturverzeichnis.....	179
	Anhang.....	193
A	Brief an Firmen.....	195
B	Vertraulichkeitserklärung.....	197
C	Leitfaden.....	199
D	Vorlage Datenblatt.....	205
E	Mail an unbekannte Beitragende.....	207
F	Auswertung Linux-Kernel-Logs.....	209
G	Verwendete Software.....	221

Abbildungen

Abb. 1:	Wachstum des Linux-Kernels von März 2005 bis Januar 2008 (Kroah-Hartman et al. 2008, S. 4) im Vergleich zu Lehman's Law.....	11
Abb. 2:	Release-Zyklus im Linux-Kernel.....	13
Abb. 3:	Rollen und Struktur einer Open-Source-Community.....	27
Abb. 4:	Vergleich von rationaler und evolutionärer Entscheidungstheorie.....	38
Abb. 5:	Wertschöpfungskette der FLOSS-Anbieter (nach Leiteritz 2004, S. 141).....	40
Abb. 6:	Open Computing.....	50
Abb. 7:	Software-Infrastruktur.....	52
Abb. 8:	Reifeprozess-Modell von Firmen in der Open-Source-Beteiligung (nach Carbone 2007, S. 5).....	54
Abb. 9:	Betriebs- und volkswirtschaftliche Auswirkungen.....	56
Abb. 10:	Wechselwirkungen im Open-Source-Ökosystem.....	61
Abb. 11:	Vergleich repräsentatives und theoretisches Sampling (nach Kruse 2006, S. 37 f.).....	77
Abb. 12:	Analyseprozess (nach Bernard 2000).....	83
Abb. 13:	Prozess der Typenbildung (nach Kelle und Kluge 1999, S. 82).....	86
Abb. 14:	Motivation zur Mitarbeit.....	92
Abb. 15:	Anzahl der Firmen, aufgeteilt nach Kontinenten.....	94
Abb. 16:	Anzahl Firmen und Codezeilen aus Europa, aufgeteilt nach Ländern.....	95
Abb. 17:	Anzahl der Firmen, nach Größe gruppiert.....	96
Abb. 18:	Anzahl Firmen nach Sektoren gruppiert.....	97
Abb. 19:	Gruppierung der Firmen nach Anzahl der Autoren.....	98
Abb. 20:	Beiträge der Firmen, gruppiert nach Anzahl der Autoren pro Firma	98
Abb. 21:	Codebeiträge zu den Modulen nach Interessengruppen.....	99
Abb. 22:	Codebeiträge zu den Modulen in Bezug zur Gesamtheit der Interessengruppe.....	100
Abb. 23:	Codebeiträge zu den Modulen nach Firmengröße.....	101
Abb. 24:	Codebeiträge zu den Modulen nach Sektoren.....	101
Abb. 25:	Codebeiträge zu den Modulen nach Sektoren.....	102
Abb. 26:	Beteiligung an der Architektur im Total pro Interessengruppe.....	103

Abb. 27:	Architekturen nach Interessengruppe in %.....	103
Abb. 28:	Architekturen nach Herkunft in % mit Lead Firma.....	104
Abb. 29:	Dimensionen für die Typenbildung.....	109
Abb. 30:	Vergleich rationale und evolutionäre Entscheidungstheorie aus Firmensicht in Anlehnung an Kuwabara (2000).....	141
Abb. 31:	Dezentralisierungsgrad (nach Malone 2004, S. 6).....	144
Abb. 32:	Erweiterte Software-Infrastruktur.....	148
Abb. 33:	Adoption durch FLOSS (nach Laisné 2008, S. 9).....	149
Abb. 34:	Rollen und Typologie von kommerziellen FLOSS-Entwicklern.....	154
Abb. 35:	Wechselwirkungen zwischen den Typen.....	158
Abb. 36:	Transitionen der Linux-Tätigkeit in Firmen.....	164
Abb. 37:	Organisation der Linux-Kernel-Entwicklung in den untersuchten Firmen.....	166
Abb. 38:	Koordination durch Distributoren.....	168
Abb. G.39:	Screenshot Zotero.....	222
Abb. G.40:	Screenshot Transcriber.....	223
Abb. G.41:	Screenshot Weft QDA.....	224

Tabellen

Tab. 1:	Übersicht zur Motivation von Individuen (nach Rossi und Bonaccorsi 2006, S. 87).....	31
Tab. 2:	Übersicht zur Motivation von Firmen (nach Dahlander und Magnusson 2005, S. 483).....	41
Tab. 3:	Übersicht der FLOSS-Geschäftsmodelle (nach Leiteritz 2004).....	42
Tab. 4:	Dauer der Interviews.....	82
Tab. 5:	Übersicht der Beiträge am Linux-Kernel im Jahr 2007.....	91
Tab. 6:	Personenjahre und Monetarisierung nach Interessengruppen.....	93
Tab. 7:	Beiträge von den Top-10-Firmen.....	93
Tab. 8:	Soziodemografischer Überblick der interviewten Linux-Kernel-Entwickler....	108
Tab. 9:	Verteilung der Interviewteilnehmenden auf die drei Typen.....	111
Tab. 10:	Problembereiche von Open-Source-Entwicklern in großen IT-Firmen.....	160
Tab. F.11:	Übersicht der Beiträge am Linux Kernel im Jahr 2007.....	209
Tab. F.12:	Übersicht der Beiträge an den einzelnen stable Releases in der Wartung.....	209
Tab. F.13:	Gruppierung nach Interessen.....	209
Tab. F.14:	Personenjahre, Monetarisierung und Aktivität nach Interessengruppen.....	209
Tab. F.15:	Interessengruppen in der Wartung.....	210
Tab. F.16:	Gruppierung nach Interessen der Sign-offs.....	210
Tab. F.17:	Interessengruppen der Sign-offs in der Wartung.....	210
Tab. F.18:	Konsolidierung der Firmen in Kontinenten.....	210
Tab. F.19:	Länderbeteiligung.....	211
Tab. F.20:	Gruppierung nach Firmengröße.....	212
Tab. F.21:	Gruppierung nach Sektoren.....	212
Tab. F.22:	Top 40 Firmen nach beigetragenen Codezeilen.....	213
Tab. F.23:	Firmenzugehörigkeit der Top 30 Entwickler.....	214
Tab. F.24:	Gruppierung der Firmen nach Anzahl Autoren.....	214
Tab. F.25:	Total Codezeilen pro Modul und Interessengruppe.....	215
Tab. F.26:	Anteil Codezeilen pro Interessengruppe und Modul.....	215

Tab. F.27: Codezeilen pro Modul in Relation zum Total der Interessengruppe.....	215
Tab. F.28: Total Codezeilen pro Modul und Kontinent.....	215
Tab. F.29: Total Codezeilen pro Modul und Firmengröße.....	216
Tab. F.30: Total Codezeilen pro Modul und Sektor.....	216
Tab. F.31: Architekturen nach Interessengruppe in Codezeilen und Prozent.....	217
Tab. F.32: Firmen mit den meisten Beiträgen pro Architektur.....	218
Tab. F.33: Top 30 Firmen im Architekturbereich.....	219

Zur Lesbarkeit des Textes

Im Fließtext werden maximal zwei Autoren in der Klammer genannt. Bei drei und mehr Autoren bzw. Herausgebern wurde nur der erste mit dem Zusatz „et al.“ erwähnt. Im Literaturverzeichnis sind jeweils alle Namen aufgeführt.

Für zahlreiche Begriffe existieren in der Informatik keine oder nur nicht gebräuchliche weibliche Schreibformen, d. h. bislang dominiert die männliche Schreibweise. Von einer Darstellung mit Schrägstrichen (z. B. „der/die Entwickler/in“) bzw. Wortkombinationen mit zweitem Großbuchstaben (z. B. „die EntwicklerInnen“) wird abgesehen, da dies die Lesbarkeit beeinträchtigt. Trotzdem hat sich der Verfasser bemüht, wo immer möglich, einen gendersensiblen Sprachstil zu wählen. Damit soll ein kleiner Beitrag zu einer genderfreundlich(er)en IT-Branche geleistet werden. Diese wird trotz zahlreicher frauenfördernder Maßnahmen noch immer von Männern dominiert. Dies gilt auch für das Engagement in der Freien- und Open-Source-Software-Community.¹

¹ Die Zahl der Frauen in der FLOSS-Community dürfte dabei noch geringer sein als diejenige der Frauen in der Informationstechnologie allgemein (Ghosh et al. 2002). Eine aktuellere Studie der EU geht dieser Tatsache noch weiter auf den Grund und entwickelt auch Empfehlungen, wie diese allseits als unbefriedigend betrachtete Tatsache geändert werden kann (Nafus et al. 2006).

1 Einführung in die Problemstellung

1.1 Ausgangslage

Freie und Open-Source-Software hat die Unternehmenswelt sowohl auf der Nachfrage- als auch auf der Angebotsseite erreicht. Spezielle Kenntnisse in der Anwendung, Entwicklung und Vermarktung von freier und quelloffener Software werden auf dem Arbeitsmarkt immer wichtiger. Open-Source-Software (OSS) ist demnach längst nicht mehr mit einer Hobby-Tätigkeit von „Geeks für Geeks“² gleichzusetzen. Zwar gibt es durchaus noch Projekte mit diesem Hintergrund, diese haben jedoch nur (noch) eine marginale Bedeutung. Mittlerweile ist allgemein anerkannt, dass Open-Source-Software zum Mainstream gehört.³ Experten gehen in ihrer Einschätzung gar so weit, dass zu Open Source weitgehend alles Wichtige gesagt sei. So wurde beispielsweise das erfolgreiche Studentenprojekt „Open-Source-Jahrbuch“⁴ letztes Jahr nach fünf Publikationsjahrgängen eingestellt. Bernd Lutterbeck, einer der Herausgeber, äußert sich in einem Interview wie folgt dazu:

„Ich glaube, dass sich die Idee Open Source in gewisser Weise erledigt hat. Erledigt hat heißt nicht, dass es überflüssig geworden ist, sondern es heißt einfach, dass der Grundgedanke überall klar ist. Es ist klar, dass dieses Modell in bestimmten Bereichen bessere Ergebnisse bringt als andere Produkte. Und dass man deswegen auch in der Praxis dazu übergegangen ist, es einfach zu machen.“
(Lutterbeck, zitiert von Rähm 2009)

Die Idee von Open-Source-Software – ursprünglich entstanden aus einer Bewegung von Computerhackern, die vornehmlich in ihrer Freizeit Software entwickelt haben – trägt noch immer den Nimbus, ein Projekt von unbezahlten Freiwilligen zu sein. Allerdings befindet sich FLOSS in einem akzelerierten Anpassungsprozess an den Markt. Diese Entwicklung vollzieht

2 In den von Eric S. Raymond betreuten sogenannten Jargon Files, einem Online-Lexikon für Begriffe rund um die Hacker-Kultur, wird ein „Geek“ beschrieben als „*a badge of pride — it's a way of declaring their independence from normal social expectations (as well as a fondness for other things like science fiction and strategy games that often go with being a hacker)*.“ http://www.catb.org/~esr/faqs/hacker-howto.html#nerd_connection [04.08.2009]

3 Eine Aussage, die das Beratungsunternehmen Gartner bereits im Jahre 2006 gemacht hat (siehe dazu z. B. <http://www.itnewsbyte.com/de/news/nws137409,.,htm> [04.08.2009]).

4 <http://www.opensourcejahrbuch.de/project> [04.08.2009]

sich entlang eines Innovationszyklus⁵, wie er z. B. von Schumpeter (1961) in der Wirtschaftswissenschaft vertreten wird. Demnach ist Freie und Open-Source-Software nach der Einführung und Durchsetzung im Markt sowie nach der anfänglich noch langsamen Verbreitung durch innovative Unternehmen jetzt im abschließenden Zyklus-Schritt und breitet sich „mit zunehmender Diffusion und Sogwirkung des Nutzenpotentials [...] immer schneller aus“ (Schumpeter 1961, S. XLIII).

Damit geht einher, dass sich schon längst nicht mehr ausschließlich Personen aus privaten Interessen an der Entwicklung von Open-Source-Software beteiligen, vielmehr hat der Anteil an Quellcode⁶ aus bezahlter, an kommerziellen Interessen orientierter Tätigkeit in den letzten Jahren stark zugenommen (Ghosh 2006). Deshalb ist heute weniger nach den Motivationen und Interessen der beitragenden Individuen, sondern vor allem nach denjenigen der Firmen zu fragen.

In dieser Entwicklung zeigen sich zwei zentrale Diskussionspunkte: 1. Die Ausweitung der OSS-Aktivitäten hin zu Kooperationen zwischen der Community⁷ und kommerziellen Firmen, und 2. der Bedarf an OSS-Spezialisten. Die Kooperation ist für den Erfolg von Open-Source-Projekten von zentraler Bedeutung.⁸ Neuartig bei Open Source ist jedoch, dass aufgrund der Lizenzbedingungen das Resultat in der Regel der Allmende, also der Öffentlichkeit zugutekommt (Lutterbeck 2005; 2006). Allerdings wurde das von Raymond (1999) als Basar bezeichnete und noch heute oft zitierte Open-Source-Entwicklungsmodell trotz zunehmender Firmenaktivitäten bei der Open-Source-Softwareentwicklung bis dato nicht um die kommerzielle Beteiligung erweitert.

5 Im wirtschaftswissenschaftlichen Verständnis dienen Innovationen der Überwindung von Ressourcenknappheit. Eine auf dieser Basis entwickelte Erfindung – wie z. B. eine kollaborativ entwickelte und frei verfügbare Software – kann erst als Innovationen bezeichnet werden, wenn sie sich im Markt durchgesetzt hat. Diese Durchsetzung findet zuerst sehr langsam statt, gefördert durch innovative Unternehmer. Mit zunehmender Diffusion und Sogwirkung verbreitet sich die Idee immer schneller, bis neuerliche Kapazitätsgrenzen erreicht sind. Diese neue Ressourcenknappheit löst einen neuen Innovationszyklus aus.

6 Quellcode oder -text nennt man die von Personen erstellten schreib- und lesbaren Algorithmen von Computerprogrammen. Diese werden in der Regel vor ihrer Verwendung in eine nur noch vom Computer interpretierbare, d. h. binäre Sprache übersetzt.

7 Unter einer „Community“ wird in diesem Text durchweg eine Netzgemeinschaft verstanden. Sie entspricht grundsätzlich dem deutschen Begriff „Gemeinschaft“, also einer Gruppe, die ein Zusammengehörigkeitsgefühl entwickelt hat.

8 Während es Kooperationen zwischen Firmen und von Firmen mit öffentlichen Institutionen wie Universitäten oder Regierungen schon länger gibt und deren Bedeutung für das Wachstum einer Volkswirtschaft erwiesen ist, entsprechen Kooperationen zwischen Firmen und Privatpersonen einem neueren Trend.

Die Beteiligung von Unternehmen an der Entwicklung von Open-Source-Software führt nicht nur zu veränderten wirtschaftlichen Beziehungen, sondern auch zu veränderten Anforderungen an Mitarbeitende im IT-Arbeitsmarkt. Am OpenWorldForum, das Anfang Dezember 2008 in Paris stattfand und auf dem Vertretungen von Firmen und Universitäten über die Zukunft von freier und quelloffener Software diskutierten, wurde geschätzt, dass bis ins Jahr 2020 bis zu 40 % aller IT-Angestellten im Bereich von Open-Source-Software tätig sein werden (Laisné 2008). Die aktuelle Studie „Potenzialanalyse im Technologieumfeld Open Source in der Hauptstadtregion Berlin“ (Fornefeld und Gasper 2009) geht aufgrund einer Online-Umfrage von annähernd 10.000 Arbeitsplätzen allein in Berlin aus, die sich direkt mit Open-Source-Software beschäftigen. Dieser Bedarf kann derzeit im Arbeitsmarkt nicht gedeckt werden. Allerdings wird beispielsweise der Stadt Berlin aufgrund der dichten Hochschullandschaft *„eine hervorragende Ausgangslage für die weitere Entwicklung des Technologiefeldes Open Source Software und den Aufbau von Kooperations- und Forschungsnetzwerken“* (Fornefeld und Gasper 2009, S. 15) attestiert. Auch mit der 2008 erfolgten Ausschreibung einer Professur für Open-Source-Software an der Universität Nürnberg-Erlangen⁹ sowie dem 2009 lancierten Studiengang „Master in Open-Source-Software“ an der Universität in Lissabon¹⁰ sind entsprechende Entwicklungen im Hochschulbereich angestoßen worden. Für die inhaltliche Gestaltung der erwähnten Bildungsmaßnahmen ist zu klären, welche Kompetenzen für die Mitarbeit an Freier und Open-Source-Software im kommerziellen Umfeld zu erwerben sind. Dies wurde in der Personalentwicklung und der Nachwuchsförderung im Open-Source-Bereich in der bisherigen wissenschaftlichen und fachlichen Diskussion jedoch noch kaum angesprochen.

1.2 Theoretische Abgrenzung

In dieser Studie geht es um Open-Source-Software und deren Integration in Unternehmensstrategien. Jedoch soll kein weiterer Beitrag zur derzeit regen Diskussion über Open-Source-Software im kommerziellen oder öffentlich-rechtlichen Umfeld geleistet werden, auch nicht zu derjenigen darüber, welches Geschäftsmodell im Zusammenhang mit Open Source geeig-

9 Siehe u.a. <http://www.heise.de/newsticker/Erste-Open-Source-Professur-ausgeschrieben--/meldung/115276> [06.08.2009].

10 Siehe dazu <http://www.linux-magazin.de/content/view/full/40591> [06.08.2009].

net ist.¹¹ Die vorliegende Arbeit befasst sich vielmehr mit den im kommerziellen Sektor angestellten Softwareentwicklern, die an Open-Source-Projekten teilnehmen und die sich dieser Tätigkeit als Hauptbestandteil ihrer Arbeit widmen. Im Speziellen wird untersucht, wie sich die firmeninterne Organisation, die betrieblichen Prozesse sowie die möglichen Spannungen in der Zusammenarbeit mit der Open-Source-Community gestalten.

1.3 Forschungsziel und erwarteter Beitrag

Empirische Forschung im Bereich von Freier und Open-Source-Software wird im Vergleich zu theoretischen Arbeiten bis dato deutlich weniger durchgeführt. Deshalb soll mit dieser Dissertation nicht nur inhaltlich, sondern auch empirisch ein Beitrag zur aktuellen Diskussion geleistet werden.

Im Zentrum der Arbeit steht die Erforschung des Tätigkeitsprofils von Open-Source-Softwareentwicklern in einem kommerziellen Kontext. Der Fokus liegt dabei auf Mitarbeitenden großer, global ausgerichteter Konzerne der Informations- und Kommunikationstechnologie (IKT), da diese quantitativ den größten Anteil an OSS-Entwicklern stellen. Zudem wurden bisherige Studien lediglich im Umfeld kleinerer Firmen durchgeführt.

Der Beitrag der Studie zur aktuellen Diskussion besteht darin, mit einer explorativen Fragestellung an den Forschungsgegenstand heranzutreten und aus den empirischen Daten Schlussfolgerungen zu generieren, die wiederum als Anregung für weitere Arbeiten dienen können.

1.4 Exkurs: „Freie“ vs. „Offene“ Software

Ob ein Projekt sich „Open-Source-Software“ bzw. „Freie Software“ nennen darf, wird ausschließlich über die verwendete Lizenz bestimmt. Sowohl die Open Source Initiative¹² (OSI)

11 Die wohl aktuellsten Daten für Deutschland liefert dazu die im Februar 2009 publizierte „Trendstudie Open Source“ (Diedrich 2009). Interessant ist die Aussage, dass sich der Einsatz von FLOSS nicht, wie bisher allgemein angenommen, „bottum up“ in der Firma entwickelt, sondern von Beginn an zur Chefsache wird. Diese Entwicklung ist ein Zeichen dafür, dass Open-Source-Software auf der Leitungsebene zunehmend strategisch verstanden wird. Als Beispiel für diese Entwicklung dient eine Einzelfallstudie über das Auswärtige Amt der Deutschen Bundesregierung (Auener 2008).

12 <http://www.opensource.org> [17.08.2009]

als auch die Free Software Foundation¹³ (FSF) – als jeweilige Repräsentanten in Form einer Non-Profit-Organisation – führen eine Liste der akzeptierten Lizenzen auf ihrer Homepage.

Zwischen den Positionen der beiden Organisationen gibt es grundsätzliche Unterschiede. Während sich Open-Source-Software an pragmatischen Ansätzen orientiert,¹⁴ versteht sich Freie Software als Ideologie.¹⁵ Letztere ist in erster Linie an der Freiheit der Software an sich interessiert. Wie diese Software entwickelt wird, ist irrelevant. Für OSS hingegen ist die Freiheit der Software lediglich Mittel zum Zweck. Sie kann das als Basar-Stil bezeichnete Entwicklungsmodell (Raymond 1999) nur durch die mittels der Lizenz garantierte Offenlegung des Quellcodes verfolgen (Moody 2008). Mittlerweile hat zwischen den Vertretern beider Seiten, die während vieler Jahre unvereinbare Positionen vertraten, eine Annäherung stattgefunden. Die Erkenntnis ist herangereift, dass beide dasselbe Ziel verfolgen, nämlich gemeinsam qualitativ hochstehende Software zu erstellen, die ohne Diskriminierungen zur Verfügung steht.¹⁶ Es wäre jedoch verfehlt, die Bedeutung der Lizenz bei Open-Source-Software zu unterschätzen. Unlängst war gar erneut in einem Online-Artikel der Financial Times Deutschland nachzulesen, OSS sei „lizenzfrei“.¹⁷ Die Lancierung der Marke „Open Source“ basierte nicht darauf, dass die Freiheit der Software infrage gestellt gewesen wäre. Vielmehr lag der Marke die Idee zugrunde, dass das Wort „free/frei“ als „gratis“ und „lizenzfrei“ missverstanden wurde, wie es der gerade genannte Artikel erneut belegt.¹⁸

Der in dieser Studie untersuchte Linux-Kernel entspricht mit der verwendeten Lizenz „GNU¹⁹ General Public Licence“ (GPL) sowohl der Open-Source- als auch der Free-Software-Defini-

13 <http://www.fsf.org/> [17.08.2009]

14 „Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.“ <http://www.opensource.org/> [12.08.2009]

15 „To use free software is to make a political and ethical choice asserting the right to learn, and share what we learn with others. Free software has become the foundation of a learning society where we share our knowledge in a way that others can build upon and enjoy.“ <http://www.fsf.org/about/what-is-free-software> [12.08.2009]

16 Siehe dazu z. B. die Keynote von Georg C.F. Greve, Free Software Foundation Europe, auf der OpenExpo im Herbst 2007 in Zürich: http://www.openexpo.ch/fileadmin/documents/2007Zuerich/01_GeorgGreve.pdf [06.08.2009] sowie auch http://www.openexpo.ch/fileadmin/documents/2007Zuerich/01_GeorgGreve.mp3 [06.08.2009].

17 Siehe http://www.ftd.de/technik/it_telekommunikation/:Unternehmen-sparen-zunehmend-bei-Lizenzgeb-%FChren-f%FCr-Software/466567.html [30.06.2009].

18 Hier zeigt sich, dass allein durch einen neuen Begriff grundsätzliche Missverständnisse nicht behoben werden können. Dies ist ein Aspekt, mit dem sich die Free Software Foundation stets kritisch auseinandergesetzt hat. Neuerdings distanziert sich die FSF wieder vom Open-Source-Begriff und sieht diesen als missbraucht und deshalb gescheitert an (siehe <http://fsfe.org/documents/whyfs.de.html> [17.08.2009]).

19 GNU ist ein Akronym für „GNU is Not Unix“.

tion. Die Linux-Community ist äußerst pragmatisch orientiert und bezeichnet Linux fast ausschließlich als Open-Source-Software. Um beiden Bewegungen gerecht zu werden, wird in dieser Arbeit in der Regel das vereinende und weit verbreitete Kürzel „FLOSS“ verwendet (Free/Libre & Open-Source-Software). Wenn eine Textstelle jedoch auf das Entwicklungsmodell fokussiert, so wird der Begriff „Open-Source-Software“ verwendet. Wenn es hingegen konkret um die Freiheit und Offenheit sowie generell um rechtliche Fragen geht, steht der Sammelbegriff „Free Software“ bzw. das deutsche Pendant „Freie Software“²⁰.

1.5 Aufbau der Studie

Als Einstieg in den theoretischen Teil wird das Projekt des Linux-Kernels kurz erläutert (Kapitel 2). Dafür wird ein Einblick in die Besonderheiten eines der erfolgreichsten, aktivsten und in Bezug auf die Anzahl der Beitragenden größten Open-Source-Projekte gewährt. Anschließend wird die Diskussion zu Open-Source-Software in einem chronologischen Abriss dargestellt, von den ersten enthusiastischen Schriften bis zu den aktuellen, weitgehend ökonomisch orientierten Beiträgen (Kapitel 3). Aus diesen Arbeiten wird die Forschungsfrage abgeleitet.

Im darauf folgenden Methodenteil werden das Design der Fallstudie (Kapitel 4) sowie die dazu gewählten quantitativen (Kapitel 5) und qualitativen methodischen Zugänge (Kapitel 6) erläutert.

Der dritte Teil, die Ergebnispräsentation, besteht aus zwei Kapiteln. Zuerst wird eine quantitative Analyse der Logfiles des Linux-Kernels von 2007 vorgestellt, in der vor allem auf die Firmenbeteiligung sowie deren Strukturen eingegangen wird (Kapitel 7). Im zweiten, umfassenderen Teil wird aus den qualitativen Interviews mit Linux-Kernel-Entwicklern im kommerziellen Umfeld eine Typologie der Entwickler erarbeitet (Kapitel 8).

Im vierten Teil werden Zusammenhänge zwischen den empirischen Erkenntnissen und dem Stand der Diskussion erläutert, dabei wird das Open-Source-Entwicklungsmodell um die Perspektive der kommerziellen Softwareentwickler ergänzt (Kapitel 9). Die Schlussfolgerungen (Kapitel 10) sowie ein Ausblick (Kapitel 11) schließen die Studie ab.

20 Für Open-Source-Software hat sich bislang noch kein deutscher Begriff etabliert. Gelegentlich wird jedoch der Begriff „quelloffene Software“ verwendet.

Teil I: Theoretischer Hintergrund

2 Der Fall Linux

2.1 Der Kernel und die Distribution

Der Begriff „Linux“ bezeichnet im Allgemeinen ein gesamtes Softwaresystem, welches einerseits das Betriebssystem wie andererseits auch eine Vielzahl an darauf basierender Software umfasst. Diese Softwares sind zum einen Anwenderprogramme, etwa ein Texteditor. Bei Linux zählen zudem zahlreiche Werkzeuge wie die Window-Oberfläche „X“ nicht zum Kern des Betriebssystems (im Gegensatz z. B. zu Windows). Deshalb wird dieses gesamte System als Distribution²¹ bezeichnet. Inhalt dieser Studie ist jedoch nur der Linux-Kernel, der als zentrale Schaltstelle im System dient. Wie Wheeler (2001) anhand der Distribution Red Hat aufzeigt, ist der Kernel die größte darin enthaltene einzelne Komponente.

Linus Torvalds, der Erfinder von Linux, hat 1991 mit ersten Arbeiten an einem UNIX-artigen²² Betriebssystem auf seinem 386er-PC begonnen, allerdings noch ohne die Absicht, ein ganzes System zu schreiben.²³ Die erste Version 0.01 (10.000 Codezeilen) wurde nach einem halben Jahr im September desselben Jahres veröffentlicht, benötigte als Basis jedoch noch das weniger freie Minix²⁴, ein ausschließlich für die Lehrtätigkeit konzipiertes System. Die Version 0.11, nach ungefähr einem Jahr Entwicklung fertiggestellt, war das erste Linux, das ohne Minix lauffähig war und somit als eigentliches Betriebssystem bezeichnet werden kann.²⁵ Die im März 1994 veröffentlichte Version 1.0 hatte bereits 175.000 Codezeilen, die Version 2.0 im Juni 1996 wuchs auf einen Umfang von 750.000 Codezeilen an. Dass heute

21 Eine umfassende Liste der Distribution ist unter <http://distrowatch.com/> [17.08.2009] ersichtlich.

22 Als „UNIX-artig“ bezeichnet man ein Betriebssystem, das sich wie ein UNIX-System verhält, ohne dass es zwingend ein solches sein muss. Eine offizielle Definition dazu existiert jedoch nicht. Einen guten Überblick zur Geschichte von UNIX bietet z. B. <http://www.levenez.com/unix/> [12.08.2009].

23 Als Torvalds zum ersten Mal mit seinem Projekt an die Öffentlichkeit trat, wies er darauf hin, es sei „*just a hobby, won't be big and professional like gnu*“ (nachzulesen in der Nachricht vom 26. August 1991 unter (<http://groups.google.com/group/comp.os.minix/msg/b813d52cbc5a044b> [12.08.2009])).

24 Das ursprünglich von Andrew S. Tanenbaum an der Amsterdamer Universität ausschließlich für Lernzwecke konzipierte und deshalb in seiner Funktionalität reduzierte UNIX-artige Betriebssystem – der Name setzt sich aus den beiden Begriffen „Minimal“ und „UNIX“ zusammen – wurde mittlerweile zu einem auch praxistauglichen freien System ausgebaut (<http://www.minix3.org/> [12.08.2009]).

25 Die kurze Entwicklungszeit wurde insbesondere dadurch ermöglicht, dass durch die Free Software Foundation und das GNU-Projekt schon zahlreiche freie Utilities zur Verfügung standen, die verwendet und integriert werden konnten. Aus Respekt und Dankbarkeit hat Torvalds Linux unter der GPL lizenziert (Torvalds 2001).

immer noch die 2er-Versionsreihe aktuell ist (Version 2.6.30, Stand 6. August 2009), zeigt, dass die Grundfunktionalität schon nach wenigen Jahren vorhanden war. Danach wurde vor allem an der Hardwareunterstützung und Stabilität gearbeitet. Ein Meilenstein war sicherlich im Jahr 1999 der Einstieg von IBM mit der Ankündigung, Linux mit hohen Geldbeträgen und großzügiger Arbeitsleistung zu unterstützen.

2.2 Der Kernel in Zahlen

Im April 2007 beinhaltet der Linux-Kernel insgesamt mehr als 21.000 Dateien mit insgesamt über 8 Millionen Codezeilen (Kroah-Hartman 2007). Die Anzahl der Codezeilen ist mittlerweile auf über 10 Millionen gewachsen²⁶ und steigt stündlich um durchschnittlich 85 Codezeilen an. Die Gesamtheit aller Codezeilen wird als Source bezeichnet und lässt sich beim Linux-Kernel in die Kategorien Kern (6 %), Treiber (30 %), Architektur (47 %), Netzwerk (5 %), Dateisysteme (6 %) und Diverses wie Dokumentation und Scripts (5 %) aufteilen (Kroah-Hartman 2007).

Im Vergleich zu 2005 hat sich die Entwicklergemeinschaft bis 2008 verdoppelt, sie umfasst mittlerweile weit über 3.500 Personen. Trotzdem wird ein Großteil des Codes von relativ wenigen Entwicklern beigesteuert. So sind beispielsweise die „Top 30“ für rund 30 % der Änderungen verantwortlich. Insgesamt rund 70 % aller Entwickler werden von gut 300 Firmen für ihre Beiträge bezahlt, wobei hier auch Non-Profit-Organisationen wie die Linux Foundation dazu gezählt werden (Kroah-Hartman et al. 2008).²⁷

Godfrey und Tu stellten anhand ihrer eigenen Analyse überrascht fest, dass der Linux-Kernel in den vergangenen Jahren eine ausgesprochen lineare Wachstumsrate aufgewiesen hatte

26 Siehe dazu <http://www.h-online.com/open/Kernel-Log-More-than-10-million-lines-of-Linux-source-files--/news/111759> [06.08.2009]. Die Anzahl hängt stark von der Methode der Zählung ab. Je nachdem, ob man Leerzeilen, Kommentare und Textfiles mitzählt oder nicht, kann die Zahl ohne weiteres um 30–40 % geringer sein. Eine eigene Zählung des aktuellen Kernels (2.6.30, Stand 6. August 2009) inkl. Leerzeilen, Kommentare und Textfiles (der Befehl dazu lautet: `find | xargs cat | wc -l`) hat eine Anzahl von 12.959.772 Zeilen Code ergeben, mit dem Programm SLOccount (<http://www.dwheeler.com/sloccount/> [06.08.2009]), ohne Zählung der Leerzeilen, Kommentare und Textfiles sind es noch 7.323.848 Zeilen. Die aktuelle „offizielle“ Zahl der Linux Foundation lautet 11.560.971 Zeilen und bezieht sich auf die Version 2.6.30 vom Juni 2009 (Kroah-Hartman et al. 2009).

27 Eine aktualisierte Studie aus dem August 2009 bestätigt die Zahlen und weist nur unwesentliche Unterschiede auf (Kroah-Hartman et al. 2009).

(Godfrey und Tu 2000). Dieses Ergebnis wurde jüngst durch Kroah-Hartman et al. (2008) bestätigt. Sie stellten während einer Zeitspanne von annähernd drei Jahren eine konstante jährliche Wachstumsrate von 10 % fest.²⁸ Dies widerspricht Lehman's Law (Lehman et al. 1997), das besagt, dass sich das Wachstum von großen Systemen wegen der direkt damit verknüpften zunehmenden Komplexität kontinuierlich verlangsamt (siehe Abb. 1).²⁹

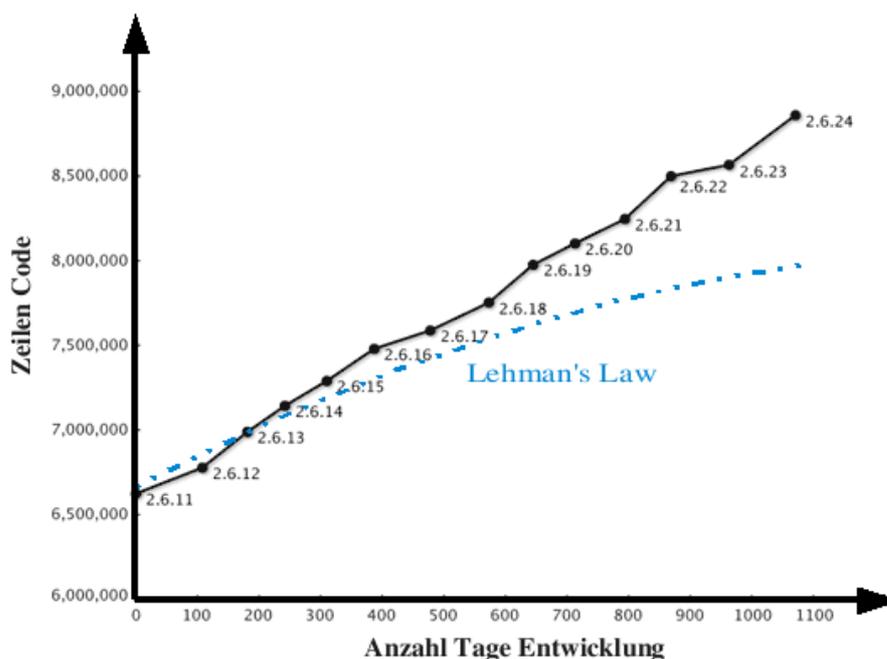


Abbildung 1: Wachstum des Linux-Kernels von März 2005 bis Januar 2008 (Kroah-Hartman et al. 2008, S. 4) im Vergleich zu Lehman's Law

Dieses lineare Wachstum kann dadurch begründet werden, dass knapp 80 % des Codes aus Hardwareunterstützung besteht,³⁰ also keine eigentliche Funktionalität beinhaltet. Durch die gewählte Architektur beim Linux-Kernel, die durch eine starke Modularisierung und eine klare Trennung der Architekturen gekennzeichnet ist, ist zudem sichergestellt, dass die Hardwareunterstützung relativ isoliert funktioniert und somit z. B. ein zusätzlicher Treiber die Komplexität nicht weiter erhöht. Auch wird die Unterstützung von alter, nur noch selten ver-

28 Diese jährliche Wachstumsrate entspricht einem stündlichen Zuwachs von 85 Zeilen Code (Kroah-Hartman 2007).

29 Lehman et al. verwenden als Metrik die Anzahl der Module. Sie haben allerdings bei einem Vergleich festgestellt, dass die verwendeten Codezeilen nicht wesentlich anders sind, aber weniger verlässliche Resultate ergeben.

30 Das Wachstum findet denn auch entsprechend bei den unterstützten Treibern und Architekturen statt, was für die zunehmende Akzeptanz und Verbreitung von Linux spricht (Godfrey und Tu 2000).

wendeter Hardware eher defensiv eliminiert, sodass auch ein beträchtlicher Anteil an nicht oder nur in sehr seltenen Fällen verwendetem Code enthalten sein dürfte (Godfrey und Tu 2000).

Hingegen lässt sich der Linux-Kernel mit dem Zipfschen Gesetz, das sich mit dem Wachstumsprozess von Systemen beschäftigt, beschreiben. Dieses Gesetz sagt aus, dass die Häufigkeit des Auftretens einer Einheit innerhalb eines Systems proportional umgekehrt zu dessen Rang ist,³¹ das heißt, dass sich die Differenzen innerhalb einer Rangordnung zunehmend verkleinern. An der ETH Zürich wurde unlängst geprüft, ob dies auch auf Linux zutrifft (Maillart et al. 2008). Anhand der Verlinkung von Software-Paketen bei der Debian-Distribution konnte belegt werden, dass Linux dem Zipfschen Gesetz entspricht. Die Studie konnte zudem belegen, dass sich diese Regelmäßigkeit erst mit der Zeit entwickelt, also wachstumsbedingt ist. Eine weitere wichtige Erkenntnis aus der Studie ist, dass die Anzahl stärker schwankt, als dass sie wächst. Dies bedeutet, dass ein Softwarepaket auch wieder aus der Distribution fallen kann, unabhängig von der Häufigkeit der Verlinkung.³²

2.3 Entwicklungsmodell des Linux-Kernels

Mit dem Beginn der 2.6er-Releases³³ des Linux-Kernels im Dezember 2001 wurde zu einem zeitbasierten Modell gewechselt, anstatt wie bisher die Release-Zyklen an einer Roadmap zu orientieren. Ab 2005 wird im Zyklus von zwei bis drei Monaten ein stabiler Release herausgegeben, der sowohl neue Funktionalitäten als auch Änderungen an der Programmierschnittstelle für andere Programme (dem sogenannten Application Programming Interface, kurz API genannt) enthalten darf. Anders als bei vielen anderen Projekten gibt es dabei keinen separaten

31 Der Linguist Zipf hat herausgefunden, dass in Texten in der Regel das häufigste Wort rund doppelt so oft vorkam wie das zweithäufigste und dreimal so oft wie das dritthäufigste. In der Folge wurde in zahlreichen Studien die Gültigkeit dieser Verteilung für die unterschiedlichsten Systeme festgestellt, so etwa für die Besucherzahlen von Webseiten oder die Größe von Städten und Firmen (Online nachzulesen unter http://www.ethlife.ethz.ch/archive_articles/090121_Zipf_nsn/index [12.08.2009]).

32 In weniger gut modulierten Systemen lassen sich Komponenten, die nicht mehr benötigt werden, sehr viel schlechter eliminieren, da die Abhängigkeiten zu anderen Komponenten nur mit viel Aufwand aufgelöst werden können. Mit dieser Tatsache sehen sich etwa die Entwickler des Betriebssystems Windows konfrontiert (Spinellis 2008).

33 Unter einem „Release“ versteht man in der Software-Entwicklung die fertig erstellte und veröffentlichte Version einer Software, die einen Entwicklungszyklus abschließt.

Entwicklungszeit³⁴ mehr, sondern es wird kontinuierlich und stetig am Hauptzweig weiterentwickelt.³⁵ Dies funktioniert so, dass zu Beginn eines Release-Zyklus ein Fenster für rund zwei Wochen geöffnet wird, in dem neue Funktionalität hinzugefügt werden darf. Die restliche Zeit dient dem Testen und dem Beheben von Fehlern. Neue Funktionalitäten oder die Unterstützung von ganz neuer Hardware wird danach nur in Ausnahmefällen und in einem kontrollierten Rahmen zugelassen. Der Release wird erst dann veröffentlicht, wenn Linus Torvalds die Qualität als befriedigend erachtet.

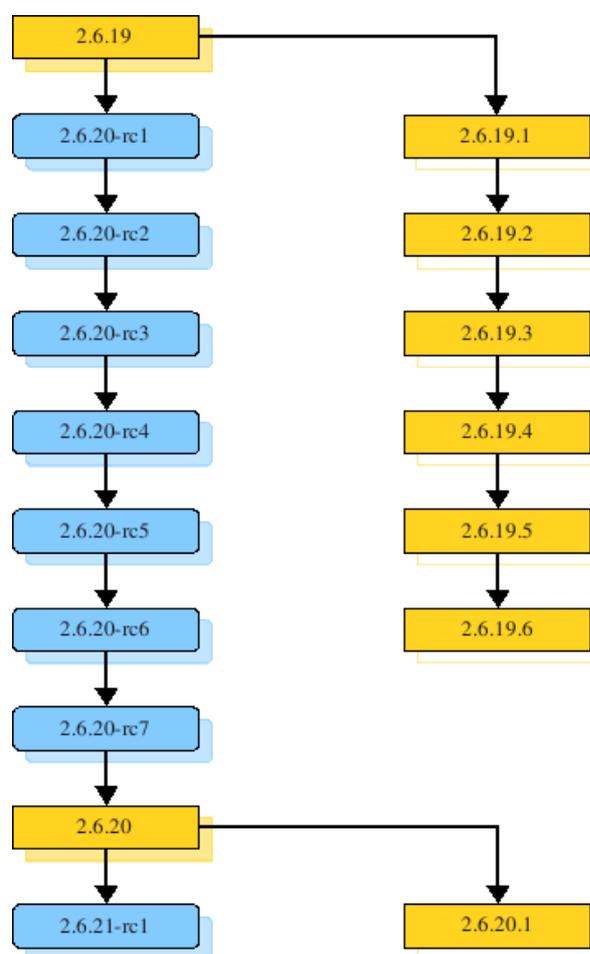


Abbildung 2: Release-Zyklus im Linux-Kernel
(nach Kroah-Hartman et al.2008, S. 2)

- 34 Es hat sich allgemein eingebürgert, ungerade Versionsnummern als Entwicklungsversionen zu bezeichnen, die bei einer gewissen Reife in eine gerade, stabile Version im Sinne eines produktiven Releases übergehen.
- 35 Insbesondere Linus Torvalds ist mit diesem Modell der zeitbasierten anstatt einer featurebasierten Versionierung sehr zufrieden, wie er auf der Mailingliste deutlich erklärte: „*The new model is so much better that it's not even worth entertaining as a theory to go back.*“ (siehe <http://article.gmane.org/gmane.linux.kernel/706594> [12.08.2009]).

Wie Kroah-Hartman et al. (2008) aufzeigen, haben in den letzten drei Jahren die Release-Zyklen zwischen 61 und 108 Tagen gedauert, mit einem Durchschnitt von 2,7 Monaten (und etwa 5.000 Patches³⁶), sie entsprechen also recht gut der Planung. Dies ist allerdings eher untypisch für Open-Source-Projekte. Der Vorteil dieses Vorgehens ist, dass recht schnell neue Hardware in einem aktuellen, stabilen Kernel unterstützt wird. Der Nachteil ist, dass nicht geplant werden kann, in welchem Release ein neues Feature enthalten sein wird.

Da Kunden von kommerziellen Dienstleistern nicht im Rhythmus von zwei bis drei Monaten einen neuen Release akzeptieren würden, liefern die Distributionen nicht jeden neuen aus. Dafür müssen die stabilen Releases noch weiter gewartet werden, damit Fehler nicht über längere Zeit in der Software bleiben und aktuelle Hardware unterstützt werden kann (siehe Abb. 2).

Von den älteren Releases werden in der Regel nur noch die zwei vorhergehenden offiziell gewartet, d. h. nach Erscheinen des 2.6.30-Releases noch die Versionen 2.6.29 und 2.6.28. Dabei ist es weitgehend Aufgabe der Firmen, insbesondere der Distributoren, diese Wartung vorzunehmen. Alan Cox hat dies anlässlich einer Präsentation über die Linux-Community auf der OpenExpo 2008 in Bern (Schweiz) prägnant erklärt:³⁷

- Übergebe die Probleme den Software- und Hardwarevendors.
- Sie haben die Experten.
- Sie haben die Kunden, die eine Lösung wünschen.
- Sie verdienen gutes Geld damit.

Firmen, insbesondere Distributoren, die noch ältere Versionen verwenden, verwalten diese in der Regel selbst, also nicht auf der offiziellen Plattform kernel.org. Da die Distributoren ihre Produkte nicht gleichzeitig herausgeben, haben diese meist auch nicht denselben Kernel integriert, wodurch es durchaus erklär- und auch vertretbar ist, dass die Wartung nicht generell auf einer zentralen Plattform stattfindet.

Um die Stabilität bei so raschen Release-Zyklen garantieren zu können, ist das Peer Reviewing äußerst wichtig. Dies wird durch eine starke Modularisierung und möglichst granulare

36 Unter einem „Patch“ versteht man im Zusammenhang mit der Linux-Kernel-Entwicklung eine Änderung des Quellcodes, wobei darin nur die geänderten Zeilen enthalten sind.

37 Präsentation: <http://www.openexpo.ch/fileadmin/documents/2008Bern/Slides/41.pdf> [17.08.2009], Video: <http://video.google.de/videoplay?docid=1893415028065590416> [17.08.2009].

Patches vereinfacht. Zudem besteht eine klare Hierarchie innerhalb der Linux-Kernel-Community, denn immer noch hat Linus Torvalds das letzte Wort, was in den Kernel aufgenommen wird. Zwischen ihm und den Tausenden von Entwicklern stehen einige Maintainer, die als Gatekeeper fungieren. Diese Funktion wurde notwendig, da mit dem stetigen Wachstum des Projekts Torvalds zum „Flaschenhals“ wurde („*Linus doesn't scale*“).

Um der Problematik zu entgehen, dass in den Wochen vor einem neuen stabilen Release keine neue Funktionalität hinzugefügt werden darf, wurde der Entwicklungszweig „Linux-Next“ kreiert. Hier kann während der Stabilisierungsphase neuer Code hinzugefügt und aufeinander abgestimmt werden, sodass bei Beginn eines neuen Release-Zyklus bereits viel neuer Code relativ reibungslos in den Hauptstrang aufgenommen werden kann. Noch weiter in die Zukunft greift der „mm-Kernel“ von Andrew Morton, der als Experimentierzweig verstanden werden kann und später nur bedingt produktiv wird.

2.4 Herausforderungen innerhalb der Community

Die stetig wachsende Codebasis sowie die steigende Zahl von Mitgliedern in der OSS-Community führen zwar nicht – wie bereits aufgezeigt – zu einer Verlangsamung des Prozesses, aber es entstehen doch zusätzliche Probleme. So äußerte sich der „zweite Mann“ hinter Linus Torvalds, Andrew Morton, bereits im Jahr 2006 öffentlich dazu, dass der Kernel immer fehlerhafter wird, weil zu schnell zu viele neue Funktionen hinzukommen. In einer strikt hierarchischen Organisation würde bei einem solchen Sachverhalt der Chef eine Weisung herausgeben. In einem Open-Source-Projekt ist dies jedoch nicht möglich. So hat der Vorschlag von Andrew Morton, neben dem Sign-off-Tag auch noch einen Reviewed-by-Tag einzuführen, zwar allgemeine Zustimmung gefunden,³⁸ in der Realität geht jedoch immer noch zu viel Code in den Kernel ein, der nie richtig kontrolliert wurde.

Ein ähnliches Problem ist der Umgang mit Fehlern. Zwar besteht eine eigene Organisation dafür, und die technische Unterstützung ist fachlich aktuell. Aber aufgrund der fehlenden Weisungsbefugnisse bleiben doch viele Fehler bei den zugeteilten Entwicklern liegen. Zahlreiche Fehler werden zudem auf inoffiziellen Wegen gemeldet und können so nicht in den offiziellen

38 Diskussion anlässlich des 2007 Kernel Summit (<http://lwn.net/Articles/248388/> [17.08.2009]).

Prozess eingehen.

Greg Kroah-Hartman, Verantwortlicher des USB-Treibers, sieht trotz der bereits großen Firmenbeteiligung noch einen vermehrten Bedarf an Unterstützung durch die Hersteller. Die weitere Verbreitung von Linux, insbesondere bei PCs und Laptops, hängt einerseits stark von der Unterstützung einer breiten Palette von Hardware ab. Wenn die Hersteller die entsprechenden Treiber nicht selbst zur Verfügung stellen und die Spezifikation nicht veröffentlichen, kann dieses Ziel kaum erreicht werden. Andererseits muss es noch mehr Software auf Linux geben. Insbesondere Businessanwendungen werden nur zögerlich portiert, weil der Markt noch zu klein ist.

Schließlich besteht ein Problem im teilweise aggressiven Schreibstil auf den Mailinglisten, der potenzielle Kernel-Entwickler abschreckt oder gar aktuelle vertreibt. James Bottomley führte anlässlich des Kernel Summit 2007 eine Session zu diesem Thema durch. Ein kurzer Auszug aus dem Protokoll.³⁹

„Flaming and generally unpleasant behavior remain a problem in the kernel community. Whenever one developer flames another - for something trivial like whitespace violations or something more substantial - he sets an example for others. The original developer may feel justified in the flaming by being ‚right‘, but those who follow may be less right while being just as inflammatory. The result is flaming by people who have never considered sending in a kernel patch. We are, says James, attracting idiots to our community by our behavior.“

Die Linux-Kernel-Community versteht Kommunikation primär als Lösungsprozess für technische Probleme sowie als Mittel zur Durchsetzung von definierten Standards. Dabei wird wenig auf die Form geachtet und meist sehr direkt kommuniziert. Die Community ist sich jedoch der sozialen Komponente der Kommunikation durchaus bewusst und ist bereit, sich selbst zu regulieren, beispielsweise indem das Problem regelmäßig thematisiert wird, etwa auf den Mailinglisten – insbesondere als direkte Reaktion auf besonders persönliche Kommentare, in der Community „flaming“ genannt – oder auf Konferenzen.

³⁹ Siehe dazu <http://lwn.net/Articles/249104/> [17.08.2009].

2.5 Zwischenfazit

Der Linux-Kernel ist das Open-Source-Projekt mit der größten Community. Die Strukturen sind während nahezu zweier Jahrzehnte gewachsen, die Hierarchien sind relativ stabil und von den Entwicklern akzeptiert. Der Fortschritt des Projekts mit einem konstanten Wachstum der Codebasis von jährlich 10 % ist außerordentlich, hat aber auch Probleme in der Qualitätssicherung zur Folge, die zudem durch die fehlende Weisungsbefugnis zur Behebung von erkannten Mängeln erschwert wird.

Die beschriebenen Studien zum Wachstum von Linux zeigen deutlich, dass das Projekt keineswegs chaotisch ist, sondern einen sehr geregelten Wachstumsprozess aufweist. Die Selbstorganisation des Projekts darf deshalb insgesamt eher als Stärke denn als Schwäche bezeichnet werden. Die Linux-Kernel-Community hat es immer wieder geschafft, die Komplexität durch intern angestoßene Veränderungen der Softwareentwicklung bzw. des Entwicklungsmodells zu meistern.

Hervorzuheben ist die erfolgreiche Synthese von kommerziellen, öffentlichen und privaten Interessen im Linux-Kernel. Allerdings wird trotz der bereits starken Firmenbeteiligung dabei noch immer ein Mangel bzw. ein Wachstumspotenzial konstatiert.

3 Open-Source-Software – Stand der Diskussion

3.1 Vorgeschichte

Die Wurzeln der Freie-Software-Bewegung sind in den späten 1950er Jahren am Massachusetts Institute of Technology (MIT) zu finden. Zwar hatten andere Universitäten auch ihre Großrechner, waren jedoch im Gegensatz zum MIT viel autoritärer organisiert⁴⁰ und deshalb als Arbeitsort für die außerhalb der sozialen Ordnung stehenden „Nerds“⁴¹ – wie etwa Richard Stallman⁴² – weniger attraktiv (Imhorst 2004). Wie der Artikel „Am Anfang war alle Software frei“ (Baumgärtel 2002) schon im Titel herausstreicht, hatte Software in den frühen Tagen der außerordentlich teuren Großrechner noch keinen Tauschwert. Dass Programme gegenseitig getauscht, begutachtet und ergänzt wurden, war im wissenschaftlichen Umfeld, wo sich die meisten Rechner befanden, denn auch weitgehend selbstverständlich und wurde informell praktiziert. Doch mit der zunehmenden Erschwinglichkeit der Server und der damit einhergehenden Verbreitung und Bedeutung vor allem in der Wirtschaft konnte die darauf laufende Software besser vermarktet werden, d. h., sie gewann wirtschaftlichen Wert und wurde infolge dessen zunehmend unter Verschluss gehalten. An den Universitäten führte dies dazu, dass die Mitarbeitenden sogenannte Non-Disclosure-Agreements⁴³ (NDA) unterschreiben mussten, und dass gute Programmierer aus den Universitäten von den Hard- und Softwareproduzenten abgeworben wurden. So sah sich Richard Stallman schon bald als „*the last true hacker*“

40 Während am MIT alle Mitarbeitenden freien Zugang zu den Terminals hatten, wachten in anderen Instituten die „Hohepriester“ (Levy 1994, S. 19) der IBM über den Zugang.

41 Auszug aus den von Raymond verwalteten Jargon Files: „*Contrary to popular myth, you don't have to be a nerd to be a hacker. It does help, however, and many hackers are in fact nerds. Being something of a social outcast helps you stay concentrated on the really important things, like thinking and hacking. [...] If you can manage to concentrate enough on hacking to be good at it and still have a life, that's fine. This is a lot easier today than it was when I was a newbie in the 1970s; mainstream culture is much friendlier to techno-nerds now. There are even growing numbers of people who realize that hackers are often high-quality lover and spouse material. If you're attracted to hacking because you don't have a life, that's OK too — at least you won't have trouble concentrating. Maybe you'll get a life later on.*“ http://catb.org/esr/faqs/hacker-howto.html#nerd_connection [17.07.2009].

42 Richard M. Stallman, auch bekannt unter seinen Initialen RMS, gründete das GNU-Projekt und die Free Software Foundation. Er ist **die** Referenzperson für den Idealismus in der Bewegung. Wegen seiner stets konsequent vertretenen, von vielen als radikal bezeichneten Position zur Freiheit von Informationen hat er allerdings auch zahlreiche Gegner.

43 Non-Disclosure-Agreements sind Vereinbarungen, die es in diesem speziellen Falle dem Softwareentwickler erlauben, den proprietären Code einzusehen. Die Weitergabe des Codes oder der konkreten Erkenntnisse aus dessen Studium sind jedoch nicht gestattet.

(Levy 1994, S. 415), da er keine Kompromisse mit der Geschäftswelt eingehen wollte.

Ähnlich wie bereits in der akademischen Welt musste im privaten Bereich auch der Homebrew Computer Club in den 1970er Jahren erkennen, dass die Kommerzialisierung von Software zunahm. Angetreten, um „*die Hackerethik weg vom Uni-Campus auf die Straße*“ (Imhorst 2004, S. 31) zu bringen, mussten die Idealisten konstatieren, dass mit dem boomenden PC-Markt auch das Hacken zunehmend zum Geschäft wurde. Diese Chance nutzten viele, um ihr Hobby zum Beruf zu machen. Sie ließen sich anstellen oder gründeten ein eigenes Unternehmen. Wie bereits am MIT führte diese Entwicklung dazu, dass Software-Codes nicht mehr geteilt wurden. Als Konsequenz daraus löste sich der Club nach kurzer Zeit im Jahr 1977 wieder auf.

Diese erste Phase der Softwareentwicklung sehen Lerner und Tirole (2002) als weitgehend informelle kollaborative Softwareentwicklung. Ein Problem dabei war, dass die Urheberrechte nicht klar geregelt waren. Dies verunmöglichte es den Urhebern, die unentgeltlich entstandene Software gegen kommerzielle Auswertung zu schützen. Darüber hinaus wurde die Bewegung durch Methoden wie das „embrace, extend & extinguish“⁴⁴ unterwandert, wie es Microsoft beispielsweise mit Kerberos und später dem Netscape-Browser erfolgreich – sowie mit Java ohne Erfolg – tat.

Deshalb dürfte der wohl bedeutendste und in seiner Tragweite nicht zu unterschätzende Beitrag von Richard Stallman für die Free-Software-Community sein, dass er zu Beginn der 1980er Jahre mit der von ihm kreierte GNU⁴⁵ General Public Licence (GPL) einen rechtlich abgesicherten Vertrag begründet hat. Das darin enthaltene, insbesondere von seinen Gegnern aufgrund der Vererbung der Rechte und Pflichten bei Anpassung und Weitergabe der Software oft als „viral“ bezeichnete Copyleft⁴⁶ – das zwingender Bestandteil einer Freien, nicht aber

44 „Embrace, extend & extinguish“ (zu Deutsch etwa „Annehmen, erweitern und auslöschen“) bedeutet, dass ein Standard zwar offiziell unterstützt („angenommen“), aber mit zusätzlichen proprietären Features erweitert wird, sodass aufgrund der eigenen Vormachtstellung im Markt ein neuer de-facto-Standard kreierte wird („ausgelöscht“). Der ursprüngliche Standard wird so unterwandert und der Konkurrenz die Kompatibilität weitgehend verunmöglicht. Diese Praxis, ursprünglich von IBM betrieben und später von Microsoft übernommen, wurde schon mehrfach erfolgreich vor amerikanischen und europäischen Gerichten beanstandet.

45 GNU ist das Akronym für „GNU is Not Unix“, dem Plan von Richard Stallman, ein komplettes UNIX-ähnliches System neu zu schreiben, das vollständig frei zur Verfügung stehen soll.

46 Im Gegensatz zum Copyright, das die Rechte des Produzenten schützt, soll das Copyleft die Rechte des Nutzers schützen. Dabei bleibt das Urheberrecht weiterhin beim Produzenten, die Rechte an der Nutzung werden jedoch dem Anwender übertragen. Er kann die Software so nutzen und auch verändern, wie es ihm beliebt.

einer Open-Source-Lizenz sein muss – stellt dabei allein sicher, dass der Hacker-Grundsatz „*information wants to be free*“ (Brand 1988, S. 201) nicht verletzt werden kann. Mit dem Copyleft wird nicht nur eine Wissens-Allmende etabliert, sondern auch ein schützender Zaun um die Allmende gezogen (O'Mahony 2003). Denn das Copyleft überlässt nicht nur zahlreiche Rechte⁴⁷ dem Nutzer, sondern es stellt auch sicher, dass die gewährten Rechte bei Weitergabe der unmodifizierten wie der modifizierten Software weiterhin gewährt werden. Richard Stallman schreibt zu seiner Motivation zur Kreation der GPL im GNU Manifesto:

„I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way.“ (Stallman 2002, S. 32)

Aus diesem Zitat wird deutlich, dass Stallman die Ideologie und Moral der von ihm hochgehaltenen Hacker-Ethik in ein juristisch fassbares, mehrseitiges und vor dem Gesetz durchsetzbares Konstrukt transferiert hat⁴⁸. Als Konsequenz daraus verpflichtet sich auch die Open-Source-Bewegung der Hacker-Ethik. Zwar versteht sie sich als pragmatisch orientiert und versucht, sich der Ideologie der Freien Software durch die unternehmensfreundlichere Bezeichnung „Open Source“ zu entziehen. Durch die Anwendung der GPL⁴⁹ gelingt ihr das jedoch aus juristischer Sicht nicht.

Während Richard Stallman und die Free Software Foundation die juristische Basis für die Open-Source-Bewegung geschaffen haben, ermöglichte erst das Aufkommen des Internets⁵⁰

47 Die Rechte beinhalten die folgenden vier Freiheiten: 1. die freie Nutzung, 2. die freie Weitergabe, 3. die erlaubte Modifizierung und 4. die freie Weitergabe von modifizierten Programmen. Die vierte Freiheit ist allerdings nur unter der Bedingung erlaubt, dass auch alle drei anderen Freiheiten weitergegeben werden.

48 Mittlerweile hat sich die Non-Profit-Organisation [gpl-violations.org](http://www.gpl-violations.org) der Durchsetzung der Lizenz angenommen. Zwar wird in erster Linie die außergerichtliche Einigung angestrebt; dass sich die GPL jedoch auch vor Gericht durchsetzen lässt, haben mehrere Prozesse belegt. Exemplarisch sei das Gerichtsurteil gegen D-Link erwähnt, nachzulesen unter http://www.gpl-violations.org/news/20060922-dlink-judgement_frankfurt.html [17.08.2009].

49 Die GPL ist weiterhin die populärste der insgesamt 73 verschiedenen Open-Source-Lizenzen, wie jüngst Perens in einem Artikel betonte (<http://itmanagement.earthweb.com/osrc/article.php/3803101/Bruce-Perens-How-Many-Open-Source-Licenses-Do-You-Need.htm> [17.08.2009]).

50 Dass ein Medium, das nicht nur senden, sondern auch empfangen kann, ein hervorragender Katalysator für die Entwicklung von Ideen und Wissen ist, hat schon Bertolt Brecht in seiner Rede über die Wirkung des Rundfunks vor über 75 Jahren erkannt: „*Der Rundfunk wäre der denkbar großartigste Kommunikationsapparat des öffentlichen Lebens, ein ungeheures Kanalsystem, das heißt, er wäre es, wenn er es verstünde, nicht nur auszusenden, sondern auch zu empfangen, also den Zuhörer nicht nur hören, sondern auch sprechen zu machen und ihn nicht zu isolieren, sondern ihn in Beziehung zu setzen.*“ (Brecht 1932/33, zitiert

in der ersten Hälfte der 1990er Jahre das Open-Source-Entwicklungsmodell in der Praxis und damit die dritte Ära der Freie-Software-Bewegung (Lerner und Tirole 2002).

3.2 Die erste Phase: Idealismus

Die Diskussion über Open-Source-Software, zumindest in Textform, beginnt wohl mit dem bahnbrechenden Essay „The Cathedral and the Bazaar“⁵¹ von Eric S. Raymond, das erstmals im März 1998⁵² in einer dem Thema Open Source gewidmeten Spezialausgabe der Online-Publikation „First Monday“ erschienen ist. Nicht ganz überraschend für einen Text, der wie so viele andere wissenschaftliche und nichtwissenschaftliche Dokumente zum Thema Freie und Open-Source-Software von einem Enthusiasten geschrieben wurde, ist seine Argumentation ideell und politisch motiviert. Lancashire fasst das Essay und damit in weiten Teilen die frühe Phase der Diskussion über Open-Source-Software wie folgt zusammen:

„In other words, although his argument is couched in the language of economics, Raymond implicitly suggests that open source development occurs outside of the market.“ (Lancashire 2001)

Raymond argumentiert in diesem sowie insbesondere auch in einem weiteren Essay zu jener Zeit („Homesteading the Noosphere“, Raymond 1999), dass Open-Source-Software auf einer Geschenkkultur (engl. „gift culture“) basiert. Er negiert somit implizit klassische ökonomische Theorien wie den „homo oeconomicus“ oder die Transaktionskostentheorie der Neuen Institutionenökonomik (wie sie von Coase 1937 eingeführt wurde) und positioniert die Open-Source-Beteiligten im Umfeld des Marxismus⁵³. Analogien bestehen demnach weniger zur

aus Brecht 1967)

51 Zwar wird das Basar-Modell als Metapher weitgehend anerkannt, es wurde jedoch auch immer wieder kritisiert. Insbesondere Bezroukov (1999a; 1999b) hat schon früh darauf hingewiesen, dass das Modell gar nicht so neu und besonders ist, sondern sich an die wissenschaftliche Praxis anlehnt.

52 In Buchform ist „The Cathedral and the Bazaar“ zusammen mit weiteren Essays von Raymond unter dem gleichnamigen Titel im Jahr 2001 erschienen.

53 Freie und Open-Source-Software wird auch von ihren Gegnern oft mit dem Attribut „kommunistisch“ bezeichnet. Stellvertretend dazu ein Auszug aus einer Kolumne von Bob Metcalfe (1999), dem Erfinder des Ethernet: „*The Open Source Movement's ideology is utopian balderdash. [...] The Open Source Movement reminds me of communism. Richard Stallman's Marx rants about the evils of the profit motive and multinational corporations. Linus Torvalds' Lenin laughs about world domination. [...] Eric Raymond breaks with Stallman, like Trotsky waiting for The People's ice pick. A Soviet Linux lies ahead, with successive five-year plans every three.*“ Interessanterweise bezeichnen sich sowohl Stallman als auch

Marktwirtschaft als zu den nicht kommerziell orientierten Nichtregierungsorganisationen.

Damit stellt Raymond auch die Open-Source-Hacker moralisch über die kommerziellen Softwareentwickler, wie bereits Levy in seinem erstmals 1984 erschienenen Buch „Hackers: Heroes of the Computer Revolution“. Himanen (2001) geht gar so weit, dass er die Hacker-Ethik im Gegensatz zur Theorie der Protestantischen Ethik von Weber (1934) als Prototyp einer Ethik für die Wissensgesellschaft sieht (Castells 2001). Diese Hacker-Ethik basiert auf dem Selbstverständnis, dass sich die Hacker selbst als Elite in einer zunehmend technologisierten Welt verstehen. In einer Art „Robin-Hood“-Kampf gegen den übermächtigen „Feind“ Kapitalismus – der sich für viele in der Form von Microsoft manifestiert – entsteht so eine Sozialromantik, die sich auch in Science-Fiction-Romanen wie etwa William Gibsons Trilogie „Sprawl“ (2000) oder Neal Stephenson's „Snow Crash“ (1995) widerspiegelt.

Die Diskussion konzentriert sich in dieser Phase auf zwei Themen: Einerseits wird das von Raymond als Basar bezeichnete Entwicklungsmodell angesprochen. Andererseits werden Erklärungen gesucht, weshalb sich Menschen in einem Projekt engagieren, das ihnen auf den ersten Blick keinen direkten Nutzen bringt.

3.2.1 Der Basar: Ein neues Entwicklungsmodell

Raymond (1999) verglich den Community-Prozess von Open-Source-Projekten mit einem Basar, wobei er sich in erster Linie an seiner Analyse von Linux und seiner eigenen Erfahrung in der Entwicklung der Software „fetchmail“ sowie an der Tradition der UNIX-Programmierung orientierte. Ganz im darwinistischen Sinne soll sich der technisch beste Code (innerhalb des Projektes oder als eigenständiges, konkurrierendes Projekt) durchsetzen. Der „Marktplatz“ ist ein öffentlich zugänglicher Basar, welcher beim Community-Prozess eine offene Internetplattform ist. Charakterisiert wird das Modell durch:

1. die Abwesenheit einer zentralen Entscheidungsinstanz,
2. kontinuierliches Designen und Debuggen des Codes,
3. die Integration der Benutzer in den Produktionsprozess und
4. die Selbstwahl der Aufgaben durch die Programmierer (Rossi 2006).

Raymond als libertär, wenn sie sich dabei auch nicht dogmatisch an die Lehren des Libertarismus halten und diesen durchaus auch unterschiedlich interpretieren (Weber 2004).

Diese vier Punkte führen unter anderem dazu, dass Open-Source-Software weniger von einzelnen Personen abhängig, stark dezentralisiert und nur sehr beschränkt planbar ist. Ghosh (1998) hat dafür alternativ zum Basar das Bild des „cooking pot market“ kreiert. Dabei fügen alle gemäß ihren Fähigkeiten Ingredienzen dem Topf bei und dürfen dafür von der Suppe essen. Der Mehrwert der Suppenkost überschreitet allerdings aufgrund der vielfältigen Beiträge die eigenen Beigaben bei Weitem. Kuwabara (2000) nennt diesen Prozess „engineering from the bottom-up“ und sieht darin eine soziale, evolutionäre Entwicklung. Damit ist auch die Dynamik eines Open-Source-Projektes angesprochen, das als sich selbst organisierendes System bezeichnet werden kann. Gemäß Probst (1987) sind damit vier Eigenschaften verbunden: Komplexität, Selbstreferenz, Redundanz und Autonomie. Die Komplexität erschwert das Beschreiben sowie Vorhersehen des Verhaltens eines Systems und begründet sich in der wechselseitigen Vernetzung, in der sich die einzelnen Teile sowie die Beziehungen untereinander ständig verändern können. Die Selbstreferenz bedeutet, dass sich das System nicht von außen steuern lässt, sondern aus sich selbst heraus handelt, und dass das eigene Verhalten Auswirkungen auf das zukünftige Handeln hat. Mit der Redundanz wird ausgesagt, dass es keine prinzipielle Trennung zwischen organisierenden, gestaltenden oder lenkenden Teilen gibt. Ein System hat dann vollständige Autonomie, wenn es die Beziehungen und Interaktionen, die es als Einheit definieren, selbst bestimmt.

Im Gegensatz zum hier beschriebenen Basar-Stil positioniert Raymond den Stil des Kathedralenbaus, wie er in der proprietären Softwareentwicklung, aber auch beim GNU-Projekt anzutreffen ist. Hier wird durch eine zentrale Steuerung die Idee eines Architekten verwirklicht. Dieser Stil führt einerseits zu einer großen Abhängigkeit (gegenüber diesem Architekten), andererseits jedoch zu einer besseren Planbarkeit. Diese duale Perspektive ist typisch für die frühe Diskussion über Open-Source-Software und auch für Raymond. Allerdings gibt es wohl nur wenige Projekte, die klar einem der beiden Modelle zugeordnet werden können (Imhorst 2004). Auch ist die mit dem Vergleich „Basar versus Kathedrale“ herangezogene Dualität in „offen ist gut“ und „proprietär ist schlecht“ zu eindimensional, denn auch proprietäre Software kann durchaus mit Basar-Attributen gekennzeichnet werden (Kuwabara 2000). Auch gibt es genügend Open-Source-Projekte, die starre und hierarchische Strukturen aufweisen und demnach der Gegenüberstellung nicht standhalten können.

Es wird immer wieder versucht, Open Source in ein allgemein gültiges Entwicklungsmodell

zu integrieren (siehe z. B. Evers 2008). Dabei wird jedoch die Betrachtung des Open-Source-Modells zu stark im Vergleich mit der Produktion von kommerzieller Software – wie etwa derjenigen von Microsoft – gesehen. Wie z. B. Perens (2007) verlässlich aufzeigt, entspricht weniger als ein Drittel der produzierten Software diesem Verständnis von kommerzieller Software, das als Standardsoftware bezeichnet wird. Der weitaus größere Anteil ist Individualsoftware, die entweder im Auftrag von Dritten oder vom Benutzer selbst „inhouse“ entwickelt wird und nie in den Verkauf gelangt. Diese Individualsoftware wird ganz anders produziert und entspricht in weiten Teilen viel eher dem Open-Source-Entwicklungsmodell. Am nächsten kommt dem Open-Source-Ansatz wohl das Extreme Programming (Beck 2000), das heute weniger als „extrem“, sondern als „agile“ Softwareentwicklung⁵⁴ in den Entwickler-Alltag eingegangen ist. Raymond fasst die Gemeinsamkeiten von Open-Source-Software und agiler Softwareentwicklung unter seiner „Regel 7“ zusammen:

„Release early, release often, and listen to your customers.“ (Raymond 1999, S. 39)

Während frühes und häufiges Veröffentlichen ungefähr gleich verstanden werden, unterscheidet sich der Einbezug der Benutzer bei Individual- und Open-Source-Software in einem Punkt erheblich: Beim Extreme Programming bzw. bei der agilen Softwareentwicklung wird zusätzlich zum Entwickler „*ein Kunde vor Ort*“ (Beck 2000, S. 61) gefordert, der im Projekt eine ganz spezifische, Know-how-vermittelnde Rolle einnimmt, selbst jedoch keinen Software-Quellcode beisteuert. Das Open-Source-Entwicklungsmodell trennt diese Rollen nicht explizit (was als Redundanz in einem sich selbst organisierenden System bezeichnet wird). Zwar kann es durchaus vorkommen, dass ein Sachverständiger „nur“ Know-how beisteuert, im Kern werden jedoch alle Beitragenden und explizit auch die Programmierer als Produzenten und als Konsumenten gesehen („Regel 1“ bei Raymond 1999).

Das Besondere des Basar-Modells sind – so Raymond – zwei Paradigmen: Erstens führt die Offenlegung und freie Verfügbarkeit des Codes in Kombination mit einer großen (Beta-Tester- und Co-Entwickler-)Gemeinschaft zu einer viel besseren Wiederverwendbarkeit (exemplarisch Weber 2000) und Qualität (Neumann 2000; Lee und Cole 2003). Raymond bringt letzteres mit dem in der Open-Source-Community häufig verwendeten Zitat auf den Punkt:

54 Das Manifest der agilen Softwareentwicklung kann unter <http://www.agilemanifesto.org/> [17.08.2009] nachgelesen werden.

„Given enough eyeballs, all bugs are shallow.“ (Raymond 1999, S. 41)

Die große Zahl Beteiligter führt zum zweiten wichtigen Paradigma: Während das anerkannte „Brooks Law“ (Brooks 1995) eine exponentielle Zunahme von Kommunikations- und Komplexitätskosten mit zunehmender Anzahl aktiver Mitglieder eines Projekts voraussagt (siehe dazu auch die Essays von DeMarco und Lister 1999), scheinen Open-Source-Projekte gemäß Raymond dem Brooksschen Gesetz nicht zu unterliegen.⁵⁵ Dies kann – wie bereits aufgezeigt – empirisch beim Linux-Kernel belegt werden (siehe dazu Kapitel 2.2). In der Regel (aber nicht zwingend) wird dies durch die Modularisierung der Programme und eine dadurch ermöglichte parallele Entwicklung erreicht (Feller und Fitzgerald 2002; Baldwin und Clark 2006), unterstützt durch das freie Lizenzmodell und das in der westlichen Gesellschaft seit Beginn der 1990er Jahre zunehmend breit verfügbare Internet. Vereinfachend wirkt sich auch aus, dass üblicherweise nur relativ wenige Entwickler für den Großteil der Arbeit verantwortlich zeichnen (Ghosh und Prakash 2000; Mockus et al. 2000). Raymond hebt demgegenüber den Führungs- und Kooperationsstil hervor, den Linus Torvalds etablieren konnte. Dieser macht eine Beteiligung am Projekt interessant und verhilft dazu, das Maximum an Leistungen im Rahmen der Voraussetzungen zu erreichen.⁵⁶ Raymond nennt diesen Stil in Anlehnung an seine anarchistischen Wurzeln⁵⁷ „*principles of shared understanding*“ (Raymond 1999, S. 64) und weist damit auf Gegensätze zu der gängigen hierarchischen Funktionalität in (kommerziellen) Organisationen hin. Eine in der Ökonomie eher gängige Bezeichnung dafür ist die „Community of Practice“ (exemplarisch Wenger 2002), wobei sich diese auf ein Zusammenwirken von mehreren Organisationseinheiten bezieht. Diese können zwar durchaus firmenübergreifend sein, sind aber nicht, wie bei Open-Source-Projekten üblich, für alle offen.

Die Etablierung einer optimalen Koordination gewinnt insbesondere für Open-Source-Projek-

55 In diesem Zusammenhang muss auch erwähnt werden, dass die meisten Open-Source-Projekte aus nur wenigen aktiven Mitgliedern bestehen (Krishnamurthy 2002). Die Ergebnisse der Studie von Krishnamurthy zeigen allerdings auch, dass die Bedeutung (Downloads, Pageviews etc.) eines Projektes in direkter Korrelation zur Anzahl der Projektmitglieder steht. Aufgrund dieser Tatsache erscheint es durchaus als berechtigt, dass sich die Betrachtungen in dieser Studie hauptsächlich auf die größeren Projekte beziehen.

56 Neben der Modularisierung von Linux (Torvalds 1999) scheint es aus heutiger Sicht sinnvoll, dass Torvalds im Jahre 1994 eine parallele Release-Struktur für den stabilen Einsatz sowie die experimentelle Entwicklung einführte, da er damit sowohl reine Benutzer als auch Hacker ansprechen konnte (Moon und Sproull 2002).

57 Raymond bezieht sich dabei insbesondere auf Kropotkin, einen der führenden russischen Anarchisten des späten 19. und frühen 20. Jahrhunderts. Kropotkin setzte sich für eine gewalt- und herrschaftsfreie Gesellschaft ein und wird als einer der wichtigsten Theoretiker des kommunistischen Anarchismus angesehen.

te enorme Bedeutung, da keine zentrale (Entscheidungs-)Instanz existiert und sich die Beitragenden ihre Aufgaben selbst auswählen bzw. zuteilen (Benkler 2003; Ettrich 2004).⁵⁸ Allerdings entwickelt sich in größeren Projekten durchaus eine Hierarchie, diese beruht aber auf Leistung und nicht auf einer formalen Steuerungs- oder gar Machtposition. Die Struktur und die darin enthaltenen Rollen einer Open-Source-Community sind in Abbildung 3 dargestellt:



Abbildung 3: Rollen und Struktur einer Open-Source-Community
(nach Nakakoji et al. 2002, S. 80)

Die Koordination wird in der Regel durch eine Projektleitung und je nach Größe durch einen inneren Zirkel (Kerngruppe), sogenannte „lieutenants“ wahrgenommen. Bekannte Formen der Projektleitung sind „benevolent dictator“⁵⁹, „rotating dictatorship“⁶⁰ und „voting committee“⁶¹ (Raymond 1999; Rossi 2006; Fogel 2006). Die Aufgabe der Projektleitenden ist es dabei nicht – wie in kommerziellen Projekten üblich –, die Ressourcen zuzuteilen, sondern sie sind hauptsächlich für die Bestimmung der allgemeinen Richtung zuständig. Die Kerngruppe trägt hauptsächlich die Verantwortung für die Koordination und die Prüfung der Codequalität. Meistens sind die Mitglieder dieser Gruppe auch berechtigt, Code direkt in das Repository⁶²

58 Diese Eigenschaften entsprechen einem sich selbst organisierenden System, wie es z. B. von Probst (1987) definiert wird.

59 Zu Deutsch „wohlwollender Diktator“; sie ist ganz typisch für Linus Torvalds und den Linux-Kernel. Nicht selten handelt es sich dabei um den ursprüngliche Projektinitiator.

60 Zu Deutsch „wechselnde Führung“, welche z. B. beim Perl-Projekt anzutreffen ist.

61 Zu Deutsch „wählendes Komitee“, was typisch für das basisdemokratische Apache-Projekt ist.

62 In der Softwareentwicklung versteht man unter einem Repository eine zentrale Lagerung des Quellcodes. Das Repository enthält in der Regel auch eine Versionierungsfunktionalität.

einzufragen. Die aktiven Entwickler unterscheiden sich von den peripheren Entwicklern, insofern sie regelmäßig Code zum Projekt beitragen. Ihre Beiträge müssen in der Regel noch von einem Mitglied der Kerngruppe angenommen werden. Während der Melder von Fehlern einem Tester entspricht und keine Kenntnisse des konkreten Quellcodes haben muss, passt derjenige, der für die Behebung der Fehler zuständig ist, das Programm an der fehlerhaften Stelle an, ohne dass beide dem Projekt Funktionalität hinzufügen würden. Der aktive Nutzer interessiert sich nicht nur für die Funktionalität der Software, sondern für das Projekt an sich. Er studiert den Quellcode oder beteiligt sich in den Foren und bei den Mailinglisten. Der passive Nutzer schließlich ist weitgehend indifferent gegenüber FLOSS, er verwendet die Software aufgrund ihrer Qualität und Funktionalität.⁶³ Grundsätzlich können jedoch beide Nutzertypen – anders als bei der proprietären Software – Kontakt mit den Entwicklern knüpfen und pflegen (Xu et al. 2005).

Die Darstellung in Kreisform in Abbildung 3 – oder in der Form einer Zwiebel, wie es viele interpretieren – unterstreicht, dass in Open-Source-Projekten nicht wie in klassischen Hierarchien von einem „oben“ und „unten“, sondern vielmehr von einem inneren und äußeren Zirkel auszugehen ist. Ausgedrückt wird damit, dass die Personen des inneren Kreises auch die Positionen der äußeren Kreise – zumindest teilweise – einnehmen. Ein Entwickler wird zum Beispiel oft auch Nutzer der Software sein. Entscheidend dabei ist, dass es keinerlei Weisungsbefugnis gibt. Der innere Zirkel hat insofern eine Machtposition, als er darüber entscheidet, wer und was aufgenommen wird. Dass er seine Machtposition jedoch nicht über Gebühr strapazieren kann, wird einerseits durch den Mechanismus des „forking“⁶⁴ sichergestellt. Andererseits lebt ein Open-Source-Projekt entscheidend von der Zahl direkt Beteiligter und Nutzer, da erst die Masse der Involvierten das Projekt sichtbar und für neue Mitglieder der Community interessant macht. Dies treibt wiederum das Projekt voran und verbessert die Reputation sowie die Karrieremöglichkeiten der aktiv Beteiligten.

Lee und Cole (2003) haben in ihrer Untersuchung des Linux-Kernels eine vereinfachte Struk-

63 Da der passive Nutzer sich nicht am Projekt beteiligt, zählt ihn z. B. Xu et al. (2005) nicht zur Community.

64 Als „forking“ wird eine Gabelung des Projektes bezeichnet, d. h. ab einem bestimmten Punkt wird der Code des Projektes in zwei voneinander unabhängige Richtungen weiterentwickelt. Da dadurch die Ressourcen auf zwei voneinander unabhängige Projekte verteilt werden, verlangsamt sich in der Regel die Weiterentwicklung und jedes einzelne der Projekte kann nicht zuletzt dadurch auch uninteressanter für aktuelle oder potenzielle Mitwirkende werden. In einer Extremsituation, in der zwei gegensätzliche Ansichten ein Projekt blockieren, kann jedoch das „forking“ auch eine entgegengesetzte Wirkung haben.

tur eingeführt, indem sie in der Kerngruppe die Projektleitung integrieren, aktive und periphere Entwickler sowie das Bug-Handling zu Beitragenden zusammenfassen und auch die Nutzer nicht in die Kategorien „aktiv“ und „passiv“ unterteilen. Da in kleineren Projekten nicht alle dieser Rollen vorkommen und mithin auch in Personalunion wahrgenommen werden können, dient dieses vereinfachte Drei-Kreise-Modell dem besseren Verständnis. Solche Modelle dienen vorwiegend der Orientierung für Außenstehende sowie für Personen bzw. Organisationen, die sich neu in einer Community engagieren möchten. Da sich in der Praxis keine formalen Status aus ihnen ableiten lassen, sind sie für aktive Community-Mitglieder von geringer Bedeutung.

3.2.2 Motivation in der Open-Source-Software-Entwicklung

Die Motivation von Entwicklern, aber auch von anderen Beitragenden wie z. B. von Testern oder Dokumentierern für eine Beteiligung an Open-Source-Projekten wird fast ausschließlich im Sinne von freiwilliger und unentgeltlicher Teilnahme diskutiert. Insbesondere Kritiker der Open-Source-Bewegung sehen dabei altruistische Motive, die aus ihrer (ökonomischen) Sicht nicht nachhaltig sind, als zentral an.

Der Begriff Altruismus ist stark mit der von Raymond (1999) propagierten Geschenkkultur von Open-Source-Software verbunden und wird u. a. auch von Kollock (1999) unterstützt, der in digitalen Gemeinschaften eine neue, effiziente Möglichkeit zur Erstellung von öffentlichen Gütern sieht. Altruistisches Verhalten ist in der Open-Source-Bewegung durchaus bedeutsam, es konnte empirisch jedoch nicht als gewichtig bestätigt werden (Hars und Ou 2002). Bekannt ist auch, dass nur diejenigen, die selbst zu einem Open-Source-Projekt beitragen, einen Nutzen über die reine Anwendung der Software hinaus ziehen können (Lerner und Tirole 2002; von Hippel und von Krogh 2003). Dies erklärt, weshalb in der Community „Trittbrettfahren“ nicht als Problem angesehen wird.⁶⁵

O'Mahony (2003) führt dazu eine neue Sichtweise ein. Während Softwareentwickler in einem

⁶⁵ Trittbrettfahrende Anwender sind zwar für die Entwicklung von Softwarecode kein Problem, denn die Masse der Anwender ist ein wichtiger Erfolgsfaktor für ein Open-Source-Projekt. Allerdings sind die trittbrettfahrenden Verwerter, die ohne eigene Beiträge den Quellcode für ein eigenes Geschäftsmodell verwenden (dies kann nur teilweise durch die Lizenz verhindert werden), bei den meisten engagierten Community-Mitgliedern nicht willkommen.

Angestelltenverhältnis nach dem Ende ihrer Anstellung sowohl das Recht als auch den Zugriff auf ihren geschriebenen Quellcode verlieren (respektive das Copyright gar nie hatten), bleibt beides bei FLOSS dem Entwickler erhalten:

„Observers of the open source phenomena questioning why contributors to community managed projects would give their work away for free have neglected to examine what is given away (code) and what is retained (rights).“ (O'Mahony 2003, S. 1180)

Die Argumentation von O'Mahony zeigt, dass die Reduzierung der Freiwilligkeit bei der Teilnahme an FLOSS-Projekten ausschließlich auf den Verzicht einer Entlohnung zu kurz greift. Das „Rätsel“, weshalb Personen bereit sind, (Opportunitäts-)Kosten auf sich zu nehmen, ohne einen direkten Nutzen davon zu haben, ergründet sich hauptsächlich auf einem eindimensionalen Verständnis des „homo oeconomicus“. Berücksichtigt man einen möglichen indirekten Nutzen im Sinne der Verhaltensökonomik⁶⁶, ist das Phänomen besser greifbar. Lerner und Tirole haben schon früh und unmissverständlich darauf hingewiesen, dass sich Open-Source-Entwickler nicht nur altruistisch, sondern durchaus rational verhalten:

„A programmer participates in a project, whether commercial or open source, only if she derives a net benefit (broadly defined) from engaging in the activity. The net benefit is equal to the immediate payoff (current benefit minus current cost) plus the delayed payoff (delayed benefit minus delayed cost).“ (Lerner und Tirole 2002, S. 212 f.)

Dieser von Lerner und Tirole erwähnte Ertrag führt unmittelbar zur Diskussion der Motivation in der Open-Source-Bewegung.

3.2.2.1 Intrinsische und extrinsische Motivation

Die Motivation zur Mitarbeit bei FLOSS ist durchaus mit gängigen Theorien der Motivationsforschung erklärbar und stellt also keinen unerklärlichen Sonderfall dar (Hertel et al. 2003).

⁶⁶ Die Verhaltensökonomik (engl. „behavioural economics“) berücksichtigt sowohl Konzepte aus der Psychologie als auch der Ökonomie. Sie wird seit vielen Jahren, u. a. führend vom Ökonomen Bruno S. Frey an der Universität in Zürich, gelehrt (stellvertretend für eine große Anzahl an Büchern und Artikel: Frey und Stutzer 2007).

Auch die oft diskutierte Frage, weshalb sich das Problem der Trittbrettfahrer von Allmenden⁶⁷ nicht stellt, ist sozialwissenschaftlich erklärbar (Kuwabara 2000). Zum einen nimmt der Wert der Software durch zusätzliche Nutzer nicht ab,⁶⁸ da sie ohne Verlust digital reproduzierbar ist. Zum anderen ist die Open-Source-Community als eine heterogene Gruppe mit unterschiedlichen Interessen anzusehen, bei der die einen mehr Interesse an der Software haben als andere. Sofern es auch Mitglieder gibt, bei denen der Nutzen die Kosten der Beteiligung übersteigt, wird das Projekt weiterentwickelt werden.

Während Lerner und Tirole (2002) bezüglich der Motivation zwischen sofortigem (Lösen eines aktuellen Problems oder Spaß an einer Tätigkeit) und verzögertem Nutzen (verbesserte Karrierechancen oder Reputation) unterscheiden, hat sich in der Open-Source-Bewegung eine Unterteilung in intrinsische und extrinsische Motivation durchgesetzt. Die einzelnen Motive sind in Tabelle 1 aufgelistet:

Intrinsische Motivation	Extrinsische Motivation
Spaß am Programmieren Altruismus Zugehörigkeitsgefühl Kampf gegen proprietäre Software	Monetäre Anreize Geringe (Opportunitäts-)Kosten Reputation unter Kollegen Zukünftige Karrierevorteile Wissenszuwachs Beiträge aus der Community Interesse an der (Informations-)Technologie Füllen einer Angebotslücke

*Tabelle 1: Übersicht zur Motivation von Individuen
(nach Rossi und Bonaccorsi 2006, S. 87)*

Intrinsische Motivation bedeutet, dass eine Aktivität um ihrer selbst willen ausgeführt wird. Gemäß Lerner und Tirole trägt sie den Nutzen in sich selbst, etwa den Spaß am Programmieren. Bei der extrinsischen Motivation hingegen steht die in Aussicht gestellte Belohnung im Zentrum. Diese ist nicht in der Aktivität selbst zu finden und wird deshalb von außen kontrolliert. In der psychologischen Literatur hat sich zudem der Begriff der internalisierten extrinsischen Motivation etabliert (Deci und Ryan 1987). Darunter versteht man extrinsische Motive, die jedoch keines direkten äußeren Anreizes bedürfen, da sie aufgrund des erwarteten unmittelbaren

67 Das „Dilemma der Allmenden“, von Hardin in seinem Essay „The Tragedy of the Commons“ (1968) beschrieben, scheint auf FLOSS nicht zuzutreffen. Dies ist dadurch zu erklären, dass durch die Digitalisierung der Allmende die Ressourcen durch die Nutzung nicht erschöpft werden.

68 Es ist eher das Gegenteil der Fall, nämlich dass zusätzliche Nutzer den Wert einer Software erhöhen. So steigt die Wahrscheinlichkeit, dass sie weiterentwickelt und breiter unterstützt wird.

telbaren Resultats einen hohen Eigennutzen haben.⁶⁹ Im Folgenden werden die einzelnen Motivations-Elemente kurz erläutert.

Spaß am Programmieren: Torvalds (2001) sieht im Spaß an der Sache und der (technologischen) Herausforderung den Kern der Motivation. Er bezieht sich dabei weniger auf Open Source als vielmehr auf die Tätigkeit des Programmierens, wie auch schon Brooks in seinem Essay „The Mythical Man-Month“ (1995) aufzeigte. Luthiger Stoll (2006) konnte bei seiner quantitativen Studie nicht nur belegen, dass Spaß für viele die Hauptmotivation für die Mitarbeit an Open-Source-Software ist. Darüber hinaus zeigte ein Vergleich mit kommerziellen Softwareentwicklern auch, dass die Arbeit in Open-Source-Projekten tatsächlich mehr Spaß macht als in einem kommerziellen Umfeld.

Zugehörigkeitsgefühl: Die Zugehörigkeit zu einer Open-Source-Gemeinschaft ist dann besonders bedeutsam, wenn sehr viel Zeit für ein Projekt aufgewendet wird und wenn bei der digitalen Zusammenarbeit kaum reale soziale Kontakte entstehen (Torvalds 2001).⁷⁰ Während Hertel et al. (2003) die Bedeutung der Identifikation anhand einer empirischen Studie des Linux-Kernels belegen konnten – und die Motivation bei Open-Source-Software als vergleichbar mit anderen freiwilligen Tätigkeiten ansehen –, konnten andere Studien dies nur in geringem Maße empirisch nachweisen (Hars und Ou 2002; Ghosh et al. 2002).

Monetäre Anreize: Geld spielt eine Rolle, da viele Entwickler ihren Beitrag innerhalb eines bezahlten Arbeitsverhältnisses leisten, sei dies regelmäßig oder nur gelegentlich (Feller und Fitzgerald 2002). Die Angaben über das Verhältnis von unbezahlten und entlohten OSS-Entwicklern divergieren in den vorhandenen Studien stark. Während Eclipse zu fast 98 % von Firmenangestellten – und davon über 60 % IBM-Mitarbeitern – entwickelt wird (Spaeth et al. 2008)⁷¹ und man beim Linux-Kernel von über 50 % ausgehen kann (Kroah-Hartman 2007;

69 „As an example, consider a person who derives considerable aesthetic pleasure from having a clean house but who does not enjoy the process of cleaning. If this person willingly chooses to clean the house, he or she would be self-determined in doing it. But the behavior would be extrinsic because it is instrumental to having a clean house, and the satisfaction is in the outcome rather than in the behavior itself. By contrast, consider another person who cleans because of a feeling that he or she has to, whether to get the approval of a business associate who will be visiting, to avoid guilt, or to satisfy a compulsion. In the case of this latter person, the extrinsically motivated behavior would be controlled.“ (Deci und Ryan 1987, S. 1034)

70 Eine Aussage, die durchaus auch kritisch bewertet werden darf, da sie letztendlich bedeutet, dass die Entwickler mehr Befriedigung bei der isolierten Arbeit am Computer als in der realen sozialen Welt finden. Zumindest auf Torvalds trifft sie jedoch während der ersten Monate – wenn nicht gar Jahre – der Entwicklung von Linux zu.

71 Eine ganz aktuelle Statistik, die vom Eclipse-Projekt selbst erstellt wurde, relativiert jedoch diese Zahlen.

Kroah-Hartman et al. 2008), nennen Hars und Ou (2002) lediglich 16 % sowie die Merit-Studie (Ghosh 2006) 33 %. Diese Differenzen resultieren hauptsächlich aus einem unterschiedlichen Projekt-Setting: Während die meisten großen, weit verbreiteten Projekte – wie Linux, der Apache-Webserver, MySQL, OpenOffice oder Eclipse – wesentlich von Firmen entwickelt werden, geschieht dies bei den zahlreichen Nischenprodukten (gelegentlich als „the long tail“⁷² bezeichnet) sowie dem Hobbybereich hauptsächlich im Privaten.

Eine andere Art des monetären Anreizes ist die Bezahlung von Individuen für ihre Leistung durch die Projektorganisation. Dies ist allerdings wenig verbreitet. Wie ein Versuch innerhalb des Debian-Projekts zeigte,⁷³ ist die Community für ein solches Vorgehen nicht zu begeistern. Zudem wirkt hier der sogenannte Crowding-Out-Effekt, d. h., dass durch die monetären Anreize die intrinsische Motivation negativ beeinträchtigt wird (Frey und Goette 1999; Fehr und Gächter 2002; Weibel et al. 2007).⁷⁴

Roberts et al. (2006) widersprechen zwar dem Verdrängungseffekt von extrinsischer gegenüber intrinsischer Motivation, stellen jedoch anhand einer empirischen Studie im Apache-Projekt eine Abhängigkeit verschiedener Motivationsfaktoren fest. So zeigen z. B. bezahlte Entwickler ein größeres Interesse an einem gewissen Status im Projekt, das Interesse als Eigenbedarf verschwindet hingegen. Dass das Status-Motiv zudem zu einer erhöhten intrinsischen Motivation führt, kann möglicherweise das Ausbleiben des Crowding-Out-Effekts erklären. Roberts et al. zeigen zudem, dass der Grund der Beteiligung an einem Open-Source-Projekt auch Einfluss auf die Menge der Codebeiträge haben kann. Beispielsweise tragen Entwickler, die aus Eigenbedarf handeln, weniger Sourcecode bei als diejenigen, die an einem Status interessiert sind.

Der Anteil der von IBM-Mitarbeitern geschriebenen Code beträgt demnach "lediglich" gut 42 %, der Anteil von Freiwilligen immerhin knapp 15 % (<http://dash.eclipse.org/dash/commits/web-app/commit-count-loc.php?sortBy=loc&show> [02.09.2009]).

72 Der Begriff „long tail“ stammt von Chris Anderson (2006) und bezeichnet die Tatsache, dass die große Masse aus Nischenprodukten besteht, für die sich der Markt im Einzelnen nicht rentiert, mit der aber in der Summe eine Rendite erzielt werden kann.

73 Um den nächsten Release trotz größerer Probleme rechtzeitig fertigzustellen, startete die Linux Distribution Debian 2006 das Experiment „Dunc-Tank“. Es sollten Schlüsselpersonen temporär gesponsert werden, was zu einer heftigen Kontroverse bis zur Boykott-Drohung in der Debian-Community geführt hatte. Das Experiment gilt als gescheitert, der angestrebte Release-Termin konnte bei weitem nicht eingehalten werden. Einen kurzen Überblick bietet ein Artikel unter <http://lwn.net/Articles/201488/> [17.08.2009].

74 Bénabou und Tirole (2000) zeigen hingegen auf, dass im Gegensatz zum Crowding-Out-Effekt durch extrinsische Anreize die intrinsische Motivation durch eine Stärkung der Selbstkompetenz erhöht werden kann.

Reputation unter Kollegen: Als ebenso bedeutsam für die unentgeltliche Mitarbeit an Open-Source-Software gilt die Anerkennung unter Kollegen (Ghosh 1998; Raymond 1999; Lerner und Tirole 2002). Da eine Open-Source-Community in der Regel meritokratisch organisiert ist (Fielding 1999), können der Status und die Reputation in der Gemeinschaft nur durch eigene, überdurchschnittliche Arbeitsleistung erreicht werden. Zwar wird von den Entwicklern Reputation durchaus als Motiv für die Mitarbeit an einem Open-Source-Projekt genannt, als alleinige Motivation kann sie jedoch nicht nachgewiesen werden (siehe z. B. Hars und Ou 2002). Das Streben nach Ansehen und auch dessen Erhaltung führen dazu, dass sich die Beteiligten weitgehend an die Regeln der Community halten und somit die Organisation vereinfacht wird, da es keiner formalen Regulierung bedarf (Markus et al. 2000). Hingegen wird es bei großen und oft kommerziell dominierten Projekten zunehmend schwieriger, in der Masse der Beteiligten an Reputation zu gewinnen. Dies führt dazu, dass in diesen Projekten das Motiv der Anerkennung unter Kollegen uninteressanter wird (Lerner und Tirole 2002). Im Gegensatz dazu ist das Motiv des zukünftigen Karrierevorteils insbesondere bei der Mitarbeit in den bekannten Open-Source-Projekten interessant. Dieses Argument wird vor allem von Lerner und Tirole (2001) angefügt und konnte auch empirisch belegt werden (Lakhani und Wolf 2003; Ghosh et al. 2002). Die Erlangung eines Jobs dank der Reputation in der Community ist jedoch in den meisten Fällen nicht der Ursprung der Beteiligung, sondern eher ein angenehmer Nebeneffekt (Ghosh 1998). Dass Reputation ganz allgemein auch einen ökonomischen Wert hat, zeigen Osterloh und Weibel (2006) in ihrem Buch „Investition Vertrauen“. Deskriptiv sowie anhand von Fallstudien zeigen sie auf, dass sich durch eine Kooperation, die auf Vertrauen anstatt auf Verträgen beruht, bessere Resultate erzielen lassen. Sie beziehen sich in ihren Ausführungen explizit auf die Erfahrungen mit Open-Source-Software und stellen diese als exemplarisch für die Bedeutung des gegenseitigen Vertrauens in einer Kooperation dar.

Wissenszuwachs: Open-Source-Software eignet sich besonders zum Erlernen und Verbessern von Programmierfähigkeiten, da der Sourcecode frei zugänglich und in der Regel auch gut strukturiert und dokumentiert ist. Der Peer-Review-Prozess, der typisch ist für Open Source, liefert denn auch ein gutes Feedback über die eigenen Kompetenzen. Vor allem für Neulinge und Studierende sind die verbesserten eigenen Fähigkeiten ein wichtiges Motiv, wie auch Bill Gates gern eingesteht:

„You’ve got to be willing to read other people’s code, then write your own, then have other people review your code. You’ve got to want to be in this incredible feedback loop where you get the world-class people to tell you what you’re doing wrong.“ (Bill Gates, zitiert in Lammers 1986, S. 83)

Dass das Erlernen, Trainieren und Verbessern von Programmierfähigkeiten tatsächlich eine wichtige Motivation zur Teilnahme an Open-Source-Projekten ist, konnte in zahlreichen empirischen Studien aufgezeigt werden (Lakhani und Wolf 2003; Ghosh et al. 2002; Hars und Ou 2002). Wichtig dabei ist auch, dass dieser Prozess aufgrund seiner Offenheit in Zusammenarbeit mit anderen und vor allem erfahrenen Entwicklern und anderer Software stattfindet, was den Reiz zur Mitarbeit weiter erhöht (Hertel et al. 2003).

Die Forschungsergebnisse von Lee und Cole (2003) zum Linux-Kernel zeigen, dass 23 % der eingesandten Beiträge nie in den offiziellen Kernel eingehen. Dies ist ein enormer Ressourcenverschleiß. Doch die Autoren argumentieren, dass aufgrund des Lerneffektes diese Beiträge durchaus keine vergebliche Mühe darstellen, sondern zu besseren zukünftigen Beiträgen führen und demnach als Lern-Investition gesehen werden müssen.

Der Lernfaktor ist ebenso für die Motivation in der peripheren Mitarbeit, die nicht aus der Programmierung der Software besteht, sehr wichtig. 98 % der Arbeit von denjenigen Personen, die in den Foren und auf den Mailinglisten Unterstützung bieten, besteht aus dem Lesen von Fragen und dem Verfassen von Kommentaren. Nur 2 % ihrer Zeit verwenden sie demnach auf ihre Antworten (Lakhani und von Hippel 2003).

Füllen einer Angebotslücke: Nicht nur die Lancierung eines Open-Source-Projekts, sondern auch dessen Weiterentwicklung basiert meist auf einem Eigenbedarf, der durch das bestehende Angebot an Open-Source-Software nicht oder nicht im gewünschten Maße abgedeckt wird. Raymond glaubt gar, dass dies für jegliche gute Software gilt:

*„Every good work of software starts by scratching a developer’s personal itch.“
(Raymond 1999, S. 32)*

Der Eigenbedarf als Motivation zur Open-Source-Beteiligung konnte empirisch bestätigt werden (Hertel et al. 2003; Hars und Ou 2002; Lakhani und Wolf 2003) und wird auch in Forschungsarbeiten meistens als Hauptgrund für die Mitarbeit genannt.

3.2.2.2 Führung über die Motivation

Angesichts dessen, dass es in Open-Source-Projekten keine formelle Weisungsbefugnis gibt, stellt sich die Frage, ob und wie sich die Beteiligten über deren Motivation beeinflussen oder gar steuern lassen. Überraschenderweise findet sich in Bezug auf Open Source dazu kaum Literatur, auch wenn sich Markus et al. (2000) diesem Thema schon sehr früh zumindest annähert haben und sich Roberts et al. (2006) in jüngerer Zeit implizit damit beschäftigen.

Markus et al. beziehen sich in ihrer Studie auf den Artikel „Management's New Paradigms“ (Drucker 1998), in dem Wissensarbeiter mit Freiwilligen gleichgestellt werden. Letzteren kann nicht befohlen, sondern sie müssen überzeugt werden. Während Drucker explizit keine Antworten auf die Frage gibt, wie sich dieses Paradigma konkret auf die Managementpraxis auswirkt, suchen Markus et al. diese Antworten in der Open-Source-Bewegung. Die Motivationsfrage ist dabei einer der untersuchten Aspekte, die Führung über die Motivation wird implizit berücksichtigt. Markus et al. kommen zu dem Schluss, dass Open Source in vielen Belangen zu spezifisch ist, als dass ihre Praktiken in eine kommerzielle Organisation transferiert werden könnten.

Roberts et al. (2006) haben untersucht, wie die unterschiedlichen Motivationen zueinander in Beziehung stehen (wie bereits im vorigen Kapitel detailliert beschrieben). Einerseits heben sie hervor, dass bezahlte Beteiligte für Open-Source-Communities aufgrund ihrer quantitativ höheren Leistung positiv bewertet werden müssen. Andererseits empfehlen sie, Community-Mitglieder aktiv über die Reputation zu motivieren, da auf diese Weise motivierte Beteiligte eine konstantere Teilnahme aufweisen.

Dass diese Lücke in der Frage der Führung über die Motivation in der OSS-Community bisher nicht angemessen mit Forschungsarbeiten gefüllt wurde, kann dahin gehend interpretiert werden, dass die Diskussion bis dato noch stark von der Ideologie geprägt war und die Fremdsteuerung nicht ins Konzept der sozialromantischen Community passt.

3.2.2.3 Opportunitätskosten

Lerner und Tirole (2002) konstatieren, dass die Motivationsdiskussion nicht ohne die Betrachtung der Kostenseite geführt werden kann. Im Allgemeinen kann ein Softwarecode heute fast ohne Fixkosten erstellt werden (Bonaccorsi und Rossi 2003). Während es früher einer teuren

Hardware-Infrastruktur bedurfte, muss heute ein Programmierer für seine Tätigkeit lediglich über einen eigenen Computer sowie einen Arbeitsplatz mit Internetzugang verfügen. Deshalb entstehen lediglich Kosten durch die aufgewendete Zeit. In einem bezahlten Arbeitsverhältnis ist dies der tatsächliche Lohn. Bei der freiwilligen Tätigkeit handelt es sich um den theoretisch entgangenen Lohn, d. h. die Opportunitätskosten⁷⁵. Entwickler könnten während der Zeit, die sie unentgeltlich für Open-Source-Software aufwenden, eine bezahlte Arbeit ausüben. Oder sie könnten in der Zeit, die im Rahmen einer bezahlten Arbeitstätigkeit aufgewendet wird, ihren Fokus auf ihre berufliche Funktion richten. Die Opportunitätskosten werden vor allem im kommerziellen Umfeld als bedeutsam angesehen, da sich daraus eine mittelbare Gewinnreduktion ergibt, wenn diese auch verzögert auftritt. Bei privatem Engagement sind die Opportunitätskosten eher theoretischer Natur und haben keine direkten monetären Auswirkungen. Lerner und Tirole (2002) weisen jedoch darauf hin, dass sich insbesondere im universitären Umfeld, aus dem sich vor allem in der Frühphase zahlreiche Open-Source-Entwickler rekrutierten, durchaus reale, wenn auch nicht direkt monetäre Opportunitätskosten ergeben, zum Beispiel durch eine verlängerte Studienzeit und den verzögerten Eintritt in den Arbeitsmarkt.

Aufgrund des offenen Quelltextes und des im Allgemeinen gut modularisierten und „sauberen“ Codes von Open-Source-Software werden die Opportunitätskosten jedoch als geringer als bei Closed-Source-Software geschätzt (Lakhani und von Hippel 2000). Somit dürfte die Einstiegshürde für Privatpersonen und Firmen niedriger sein, was sich förderlich und motivierend auf ihr Engagement im Open-Source-Bereich auswirken kann.

3.2.2.4 Rationale und evolutionäre Entscheidungstheorie

Die gängige Motivationsdiskussion, die bei FLOSS im Zentrum steht, orientiert sich an der Theorie der rationalen Entscheidung (engl. „rational choice theory“, exemplarisch Elster 1986). Diese geht davon aus, dass Subjekte ihr Verhalten auf die Maximierung ihres individuellen Nutzens aufgrund einer subjektiven Situationsbetrachtung ausrichten, der Inhalt – im vorliegenden Falle die Software – ist dabei oft nur Mittel zum Zweck. Bei Kenntnis der Präfe-

75 Unter Opportunitätskosten versteht man denjenigen Nutzen, auf den man verzichtet und den man erreichen könnte, wenn man anstatt der ausgeführten Aktivität einer anderen (bezahlten oder unbezahlten) Beschäftigung nachgehen würde.

renzen des Subjekts ist somit auch sein Verhalten vorhersagbar. Nicht berücksichtigt wird in diesem Modell, welche Auswirkungen das eigene Handeln auf die Umwelt hat und welche Wechselwirkungen mit dieser bestehen.⁷⁶ An dieser Kritik setzt Kuwabara (2000) an und führt ein evolutionäres Modell in die Diskussion zur Motivation in Open Source ein, das sich an den Erkenntnissen der evolutionären Spieltheorie (Smith 1982) orientiert.

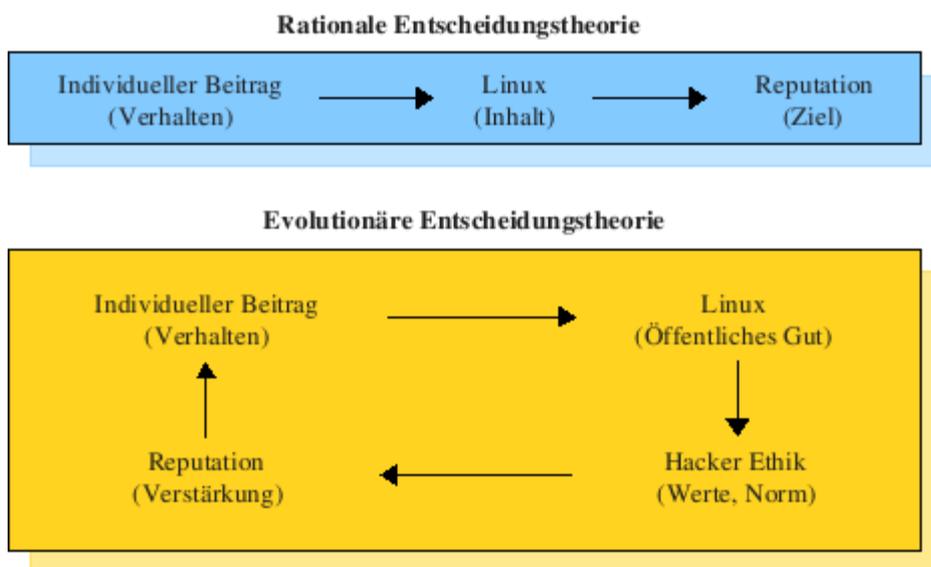


Abbildung 4: Vergleich von rationaler und evolutionärer Entscheidungstheorie (nach Kuwabara 2000)

Bei der evolutionären Entscheidungstheorie (siehe Abb. 4) wird das Verhalten nicht ausschließlich durch ein rationales Kalkül der individuellen Nutzenmaximierung bestimmt, sondern Entscheidungen werden als das Ergebnis eines kulturellen Evolutionsprozesses verstanden. Durch diesen zusätzlichen, dynamischen Einfluss ist auch das Verhalten einer Person nicht mehr vorhersagbar, und es sind analog zur Spieltheorie Annahmen über das Verhalten anderer Beteiligter zu treffen.

Dieses Konzept berücksichtigt, dass soziale Einflüsse – wie im Falle von Open-Source-Software z. B. die Werte und Normen der Community – Teil des zirkulären Entscheidungsprozesses und somit keine Konstanten mehr sind. Es kann insofern nicht mehr von einem Ziel gesprochen werden, da es keinen Endpunkt zu erreichen gilt, sondern jeder Abschluss eines Zy-

⁷⁶ Im Gegensatz zum allgemeinen Verständnis in der Ökonomie wird diesen Aspekten in der Soziologie besonders in jüngerer Zeit vermehrt Beachtung geschenkt (siehe dazu z. B. Becker 1993 oder Esser 2000).

klus ist gleichzeitig die Basis für den Beginn eines neuen. Kuwabara nennt dies einen Verstärker, und den gesamten Zyklus bezeichnet er als positiven Feedback-Loop. Dieser bewegt einerseits aktive Community-Mitglieder dazu, dem Projekt treu zu bleiben. Andererseits signalisiert der Erfolg des Projekts auch Außenstehenden, dass eine Beteiligung attraktiv ist.

3.2.3 Zwischenfazit

In der frühen Phase der Diskussion des Phänomens FLOSS, die vorwiegend von Beteiligten oder der Bewegung ideell nahestehenden Personen geführt wird, steht die kollaborative Produktion der Software sowie die individuelle Motivation zur Mitarbeit im Vordergrund. FLOSS wird als soziales Phänomen (Kuwabara 2000) und als Sonderfall verstanden, der sich an den Gesetzen der nicht kommerziellen Nichtregierungsorganisationen orientiert.⁷⁷ Eine ökonomische Bedeutung wird zwar nicht negiert – um freie Software auch für Firmen attraktiv zu machen, wurde die Open-Source-Initiative seinerzeit lanciert –, aber es wird ihr eher ein Sonderstatus und eine Nischenberechtigung zugesprochen. Die Motivation zur Beteiligung wird ausschließlich aus einer individuellen Sicht betrachtet, je nach Studienergebnissen stehen eher intrinsische oder extrinsische Motive im Vordergrund. Zwar wird dieses sozialromantische Verständnis eines Sonderfalls von einigen Studien widerlegt, die sowohl das Entwicklungsmodell als auch die Motivation zur Mitarbeit durchaus kompatibel zu bestehenden Theorien sehen. Diese Sicht ist in der frühen Phase jedoch nicht mehrheitsfähig bzw. wird von den Hauptakteuren weitgehend ausgeblendet.

Um das Jahr 2000 begann das Open-Source-Engagement von Unternehmen.⁷⁸ So ist es nachvollziehbar, dass die Autoren in der frühen Phase der Auseinandersetzung mit FLOSS diese noch weitgehend im Sinne einer „Geschenkultur“ betrachteten und der Eindruck entstand, dass FLOSS außerhalb des kapitalistischen Marktes funktioniere (Perens 2007). Die diskutierten und gelebten Normen und Werte der ersten Phase sind für viele in FLOSS engagierte Per-

77 Diese Tendenz zeigt sich mithin auch auf einer Plattform, die sich an die Philosophie der FLOSS-Bewegung anlehnt (<http://www.oekonux.de/> [17.08.2009]). Unter anderem soll hier diskutiert werden, „*ob die Prinzipien der Entwicklung Freier Software eine neue Ökonomie begründen können, die als Grundlage für eine neue Gesellschaft dienen könnte*“ (aus der Kurzbeschreibung zum Projekt auf der Homepage, zuletzt zugegriffen am 28.05.2009).

78 IBM zum Beispiel hat als erstes großes und im Markt etabliertes Unternehmen seine Ankündigung zur milliardenschweren Unterstützung von Linux im Jahre 1999 getätigt, also erst nach den Publikationen von Raymond.

sonen auch heute noch bedeutsam. Sie sind jedoch eher eindimensional; und es wird in den nachfolgenden Kapiteln aufzuzeigen sein, wie eine differenziertere Diskussion, die auch die ökonomische Perspektive berücksichtigt, geführt werden kann.

3.3 Die zweite Phase: Pragmatismus

FLOSS wurde zu Beginn und bis dato oft als Gratissoftware verstanden. Zwar konnte sich die Marke „Open-Source-Software“ anstelle der „Free Software“ etablieren.⁷⁹ Zumindest in einschlägigen (Business-)Kreisen wurde jedoch auch erkannt, dass dieser Wechsel nicht nur monetär zu betrachten ist, sondern auch Fragen zu den Rechten und zum Modell der kollaborativen Entwicklung aufwirft. Ghosh (1998) wies bereits vor zehn Jahren darauf hin, dass sich aus ökonomischer Sicht der Wert eines Produkts nicht in erster Linie über den Preis definiert, sondern über den Nutzen für Produzent und Konsument.⁸⁰ Während der Nutzen des Konsumenten einfach definiert werden kann – eine kostengünstige, in Bezug auf Funktionalität und Stabilität kompetitive Software –, ist dies auf der Produzentenseite schwieriger. Hier wurden bisher über einen „trial-and-error“-Prozess Geschäftsmodelle bezüglich Nachhaltigkeit getestet.⁸¹

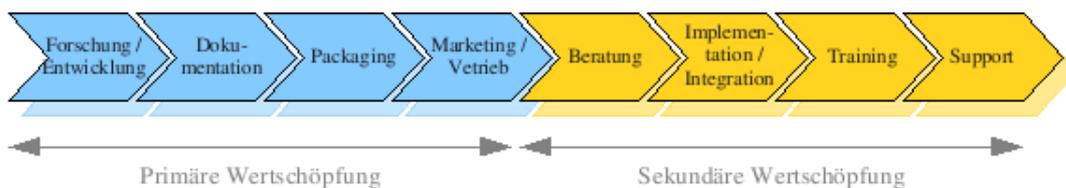


Abbildung 5: Wertschöpfungskette der FLOSS-Anbieter (nach Leiteritz 2004, S. 141)

79 Richard Stallman und die Free Software Foundation betonen, dass sich das Wort Free nicht auf den Preis, sondern auf die Freiheit bezieht. In diesem Sinne hat sich das Verständnis durchgesetzt, dass es nicht um „free beer“ (Gratisbier), sondern um „free speech“ (freie Rede) geht.

80 Hinzu kommt, dass in der realen Welt die Rollen von Produzenten und Konsumenten klar aufgeteilt sind. Allerdings gibt es – wie die Wortbildung „Prosument“ verdeutlicht – einen Trend zu einer Vermischung, was die Rollenteilung in der digitalen Welt erschwert. Gerade bei FLOSS kann jeder Konsument zum (Mit-)Produzenten werden.

81 So hat sich zum Beispiel der Verkauf von vorgefertigten, benutzerfreundlichen Linux-Distributionen als erstes Geschäftsmodell etabliert. Anbieter wie Red Hat oder Suse Linux erkannten jedoch bald, dass sich die Distribution mit dem zunehmenden Aufkommen von breitbandigem Internet kommerziell nicht lohnt. Deshalb bieten sie heute erfolgreich Dienstleistungen rund um ihre Distribution an.

Leiteritz (2004) erklärt anhand der Wertschöpfungskette einer Software, wie und wo Profit mit einem kostenlosen Produkt gemacht werden kann (siehe Abb. 5). Je nach Geschäftsmodell können eines oder mehrere dieser Wertschöpfungsglieder als Gewinnquelle herangezogen werden. Während bei der Kontrolle der gesamten Kette – wie es bei proprietärer Software weitgehend üblich ist – sowohl der Ressourcenverbrauch als auch der Gewinn sehr hoch sein können, kann bei FLOSS-Anbietern der Ressourcenverbrauch durch die Konzentration auf nur wenige oder gar ein einziges Glied der Wertschöpfungskette deutlich reduziert werden. Nachteilig dabei ist, dass die Gewinnerwartung weniger hoch sein dürfte. Aufgrund des geringeren Ressourcenverbrauchs wird das FLOSS-Modell besonders interessant für kleine und mittlere Unternehmen.

3.3.1 Motivation aus Firmenperspektive

Im Unterschied zur Unterteilung der Motivation in intrinsisch und extrinsisch, wie sie für die Entwicklerperspektive nachgezeichnet wurde, führen Feller und Fitzgerald (2002) aus Firmenperspektive drei Kategorien für die Motivation ein; die einzelnen Motive, gruppiert in diesen drei Kategorien, sind in Tabelle 2 aufgelistet.

Wirtschaftlich
Software-Entwicklung beschleunigen Wettbewerbsfähigkeit erhöhen Konkrete Geschäftsmodelle realisieren Entwicklungskosten senken
Sozial
Code mit der Community teilen Unterstützung der Idee frei verfügbarer Software
Technologisch
Feedback ausnutzen Fachlicher Support in früher Entwicklungsphase erhalten (Offene) Standards unterstützen

*Tabelle 2: Übersicht zur Motivation von Firmen
(nach Dahlander und Magnusson 2005, S. 483)*

Die Ergebnisse einer empirischen Studie zeigen, dass die wirtschaftlichen Motive für Firmen entscheidend sind, um in der Open-Source-Community aktiv zu sein (Bonaccorsi und Rossi 2003). Die Überzeugung, dass Information frei geteilt werden soll, mag bei der Firmengründung eine wichtige Rolle spielen (Lerner und Tirole 2002; Feller und Fitzgerald 2002). Im

wirtschaftlichen Alltag werden die sozialen Motive jedoch meist unbedeutend.

Gerade in einem (Quasi-)Monopol, wie es z. B. Microsoft und Oracle für ihre Produkte durchgesetzt haben, kann eine Firma durch die Offenlegung des Quellcodes⁸² mitunter rasch eine beträchtliche Verbreitung und Sichtbarkeit in der IT-Landschaft erlangen und wird dadurch für neue und talentierte Entwickler attraktiv (West 2003). Als Folge dessen kann die Entwicklung beschleunigt und eine größere Innovationskraft entwickelt werden (Lerner und Tirole 2002). Daraus ergibt sich eine indirekt verbesserte Wettbewerbsfähigkeit des Projekts, wie dies beispielsweise der Webbrowser Firefox oder die Office-Suite OpenOffice in Konkurrenz zum Giganten Microsoft gezeigt haben.

Allerdings ist der wichtigste Grund für die aktive Beteiligung eines Unternehmens an einem FLOSS-Projekt ein konkretes Geschäftsmodell, das die Besonderheiten der freien Software berücksichtigt, wie Raymond (1999) bereits vor zehn Jahren aufzeigte.

3.3.2 Neue Geschäftsmodelle

Da sich mit Lizenzen – dem traditionellen Geschäftsmodell mit Software – bei Open-Source-Software kein Gewinn erzielen lässt, müssen Firmen andere Möglichkeiten zum Wirtschaften finden. Leiteritz (2004) unterscheidet dabei zwischen Produkt- und Dienstleistungs-Geschäftsmodellen (siehe Tabelle 3), wobei in der Realität oft beide Ansätze kombiniert werden.

Produkt-Geschäftsmodelle	Dienstleistungs-Geschäftsmodelle
Distributor Applikationsanbieter Appliance-Hersteller	Support/Wartung Beratung Implementation/Integration Schulung/Dokumentation Mediation

Tabelle 3: Übersicht der FLOSS-Geschäftsmodelle (nach Leiteritz 2004)

82 Wie West (2003) erläutert, gibt es neben FLOSS-Projekten, die durch Individuen und Gemeinschaften lanciert und gefördert werden, auch diejenigen Projekte, die zuerst proprietär durch eine Firma entwickelt und erst später freigegeben wurden. Während die Motivation zur Freigabe durchaus denjenigen zum Anschluß an ein bestehendes Projekt entsprechen, unterscheidet sich der Community-Prozess doch zum Teil erheblich (West und O'Mahony 2005), insbesondere die Bildung einer Entwickler-Community mit einer bestehenden Codebasis stellt sich als erheblich problematischer dar.

Bei den großen, etablierten IT-Unternehmen wie etwa IBM⁸³, HP und Sun, die alle FLOSS aktiv unterstützen, bestehen die Geschäftsmodelle in einer Kombination verschiedener Ansätze. Insofern muss hier nicht von einem reinen, sondern von einem hybriden Open-Source-Geschäftsmodell gesprochen werden. Es ist nicht klar ersichtlich, in welchem Geschäftsbereich Gewinn aus FLOSS erwirtschaftet wird. Es wird allerdings von einem Engagement aufgrund einer direkten Gewinnerwartung, z. B. aus dem Verkauf von Hardware, auf der Linux läuft, ausgegangen (Leiteritz 2004).

3.3.2.1 Produkt-Geschäftsmodelle

Die drei in Tabelle 3 aufgeführten Produkt-Geschäftsmodelle werden hier in der Folge kurz erläutert:

Distributor: Vor allem im Zusammenhang mit Linux sind in der zweiten Hälfte der 1990er Jahre zahlreiche Firmen entstanden, die aus den vielen, dezentral vorliegenden Komponenten ein lauffähiges, anwendbares Software-System zusammenstellten (z. B. Red Hat oder Suse) und darauf einen „Brand“ aufgebaut haben (Young 1999). Das ursprüngliche Geschäftsmodell mit dem Verkauf von Datenträgern mit der installierbaren Software war in erster Linie auf Privatanwender ausgerichtet. Inzwischen ist es jedoch unbedeutend, da sich wegen der Verfügbarkeit von Breitband-Internetverbindungen mittlerweile keine nennenswerte Rendite mehr erzielen lässt. Die Distributoren haben deshalb einerseits ihren Kunden Mehrwert durch zusätzliche, proprietäre Software (meist für die Administration des Systems) angeboten und sich zunehmend auf Geschäftskunden konzentriert. Andererseits – und dies scheint der erfolgreiche Weg angesichts der zunehmenden Öffnung der proprietären Software – bieten die Distributoren Dienstleistungen wie Abonnements oder Garantien an. Das konkrete Geschäftsmodell der Distributoren setzt sich dabei aus der primären Wertschöpfung und dem Support zusammen (Leiteritz 2004). Dieses Modell entspricht dem proprietären Modell, jedoch mit dem entscheidenden Unterschied, dass ein großer Teil der Systemkomponenten unter einer freien Lizenz steht (Rosenberg 2000). Die Distributoren arbeiten nicht nur mit der Community zu-

83 IBM z. B. erwirtschaftet seit Jahren stets weniger Gewinn mit dem Hardwaregeschäft und konzentriert sich dafür auf lukrative IT-Dienstleistungen. So kann IBM auch in der aktuellen Wirtschaftskrise ihren Gewinn auf hohem Niveau halten, wie unlängst ein Artikel in der Neuen Zürcher Zeitung festhielt (http://www.nzz.ch/nachrichten/wirtschaft/aktuell/ibm_mit_krftigem_gewinn_1.3078870.html [17.08.2009]).

sammen, sondern auch mit den diversen Hard- und Softwareherstellern, um die Unterstützung möglichst vieler Plattformen garantieren zu können. Dies ist vermutlich der größte Mehrwert, den die Distributoren bieten.

Applikationsanbieter: Erfolgreiche Geschäftsmodelle mit dem Angebot von Applikationen sind eher selten. Gewinne können einerseits durch die Segmentierung des Marktes mit unterschiedlichen Lizenzen,⁸⁴ sogenannten „Duallizenzen“ (Valimaki 2003), generiert werden. Bekanntestes Beispiel dafür ist der schwedische Datenbankanbieter MySQL AB. Die Firma verfolgt die Strategie, dass sie den weniger lukrativen (Privat-)Markt mit einer freien Version bedient, die allerdings in ihrer Funktionalität reduziert ist. Dennoch erreicht sie damit einen Marketingeffekt. Da aus juristischen Gründen in aller Regel das Verwertungsrecht bei der jeweiligen Firma liegt, sind solche Projekte für Freiwillige eher uninteressant. Deshalb kann niemand von diesem Pool profitieren, der sein Geschäftsmodell auf Duallizenzen aufbaut. Dies führt dazu, dass die gesamte Wertschöpfungskette vom jeweiligen Unternehmen getragen werden muss.

Ein weiteres Modell besteht darin, Einnahmen, die den eigenen Aufwand für das Open-Source-Projekt übersteigen (Raymond 1999), durch Komplementärsoftware⁸⁵ respektive durch das kommerzielle Abdecken der sekundären Wertschöpfungskette zu generieren (Raymond 1999). Feller und Fitzgerald (2002) nennen dies ein hybrides Modell⁸⁶ im Gegensatz zum „pure-play“-Open-Source des Duallicensing. Während beim hybriden Modell analog zur proprietären Softwareentwicklung keine direkte Konkurrenz innerhalb des Produktes entstehen kann, entscheidet im „pure-play“ die konkrete Leistung über den Erfolg. Weitaus häufiger sind jedoch FLOSS-Applikationsanbieter, die ihre Einnahmen nicht direkt aus der Software beziehen, sondern diese aus strategischen Gründen frei anbieten. Sie verfolgen dabei das Ziel, Konkurrenten zu schwächen bzw. eigene Abhängigkeiten zu reduzieren, wie dies z. B. Sun mit der freien Office-Suite OpenOffice gegenüber Microsoft tut.

84 Dieses Prinzip wurde nicht von Open Source erfunden. Es war schon immer Ziel, den Markt in Segmente zu unterteilen, um dann pro Segment u. a. eine optimale Preispolitik gestalten zu können.

85 Die Projektplanungs-Software Open Workbench der Firma CA kann z. B. durch die Software Clarity derselben Firma erweitert werden, wenn man eine netzwerkbasierete Lösung benötigt.

86 Dieses Modell wird gelegentlich auch „open core“ genannt, d. h., der Kern ist frei, darum herum wird aber proprietäre, kommerzielle Software angeboten. (Siehe dazu das Beispiel von Wireshark, wie im Artikel <http://www.heise.de/open/Wireshark-ein-Business-Modell-steht-Kopf--/artikel/141265> [17.08.2009] von Heise erläutert wird.)

Appliance-Hersteller: Das Angebot von Hardware-Software-Betriebssystem-Kombinationen aus einer Hand (sogenannte „Appliances“ sowie „Embedded“-Systeme) bietet eine zusätzliche Motivation für die Partizipation an Open Source. Da sich der Preis des Pakets in erster Linie aufgrund des versprochenen Nutzens bestimmt, kann durch die Verwendung von kostenlosen Teilen, z. B. dem Betriebssystem Linux, eine höhere Marge erreicht werden. Das Geschäftsmodell setzt sich dabei aus der primären Wertschöpfung und dem Support zusammen. Problematisch bei vielen Appliance-Anbietern ist, dass sie oft Software integrieren, die unter der GPL lizenziert wurde und somit eigene Anpassungen veröffentlichen müssten, dies aber aus Nachlässigkeit oder der Erwartung strategischer Vorteile nicht tun. Um diese Prozesse zu verhindern, wurde die Non-Profit-Organisation gpl-violations.org gegründet, die auch bereits einige Erfolge zu verzeichnen hat.⁸⁷

3.3.2.2 Dienstleistungs-Geschäftsmodelle

Support/Wartung, Beratung, Implementation/Integration, Schulung/Dokumentation: Drei Vorwürfe an OSS bieten gleichzeitig die Chance, auch ohne eigene Software-Produkte Gewinn erwirtschaften zu können. Die Vorwürfe sind: Erstens ist die Software oft nicht „out-of-the-box“ einsetzbar, sodass bei der Installation und beim Betrieb entweder Know-how aufgebaut oder eingekauft werden muss.⁸⁸ Zweitens gibt die Open-Source-Gemeinschaft keine Gewähr auf einen Support, dieser ist aber für manche Anwender unabdingbar. Und drittens fehlt eine Herstellergarantie.

Deshalb bietet sich die Chance, neben der Beratung, dem klassischen Support und der Wartung auch die Integration in bestehende Software-Systeme als wichtiges Betätigungsfeld aufzubauen. Dazu gehören Schulung und Training sowie die Publikation von Handbüchern. Während sich das Geschäftsmodell kaum von herkömmlichen IT-Dienstleistungen unterschei-

87 Die Organisation mahnt Unternehmen, die die GPL verletzen, vorerst ab und fordert sie auf, sich gemäß den Lizenzvereinbarungen zu verhalten. Da diese Abmahnungen sehr oft zu einem Umdenken und Einlenken bei den betroffenen Unternehmen führen, geraten nur sehr wenige Fälle vor Gericht. Wie z. B. der erfolgreiche Prozess gegen D-Link aufzeigt, steht auch bei einer Verurteilung nicht eine Buße im Vordergrund, sondern die Einhaltung der Lizenz (siehe dazu das Gerichtsurteil unter http://www.jbb.de/urteil_lg_frankfurt_gpl.pdf [30.08.2009]).

88 Peter Deutsch, Verfasser der freien Software Ghostscript und Gründer einer der ersten Open-Source-Firmen, kam in seiner Rede auf der ersten „Freely Redistributable Software Conference“ (1996) zu dem Schluss, dass der Grundsatz „release early, release often“ nicht zuletzt deshalb so verbreitet war, weil er aufgrund der noch unfertigen Software ein Dienstleistungsgeschäft ermöglicht.

det, kommt hier ein zentrales Element hinzu: Es gibt keine Anbieterbindung. Dies führt dazu, dass auf der Anbieterseite der Konkurrenzdruck verstärkt wird und sich die Anbieter an den Kundenwünschen und -bedürfnissen orientieren müssen.

Der Gewinn wird hauptsächlich aus der sekundären Wertschöpfung generiert oder allenfalls durch Entwicklung. Der Umsatz entsteht dabei in der Regel aus verrechneten Stundensätzen. Deshalb hat der Anbieter keinen direkten monetären Nutzen aus der Verwendung von FLOSS. Die Einsparung der Lizenzen fallen gänzlich bei der Kundschaft an. Der Anbieter hat jedoch aufgrund der freien Lizenz einen viel größeren Handlungsspielraum und mehr Einflussmöglichkeiten auf die Software. Aus der Beteiligung an FLOSS kann der Anbieter nicht nur Know-how aufbauen, sondern seine Beiträge dienen gleichzeitig als Leistungsausweis.

Mediation: Da die Open-Source-Welt sehr heterogen, nicht organisiert und oft schlecht vermarktet ist, ist es die Aufgabe von Mediatoren, Anbieter und Nutzer im Sinne eines Marktplatzes zusammenzubringen (Grassmuck 2002). Der wohl bekannteste Mediator ist die Plattform SourceForge, aber auch die Firma Optaros hat sich hier weltweit etabliert. Mediatoren leben von ihrer Größe, die auch zu einem Lock-In führen kann. Da zwischen FLOSS-Anbieter und -Nutzer meist kein Geld fließt, kann sich der Mediator auch nicht über Provisionen finanzieren. Der Gewinn muss deshalb über einen anderen Kanal wie z. B. die Platzierung von Werbebannern auf Internetportalen fließen oder indirekt durch Folgegeschäfte aufgrund eines Marketingeffektes erwirtschaftet werden.

3.3.3 Eine neue Form der Organisation

O'Mahony (2002) stellte aufgrund von 75 halbstandardisierten Interviews mit Open-Source-Entwicklern fest, dass Firmen, die sich an FLOSS beteiligen, ihr Verhalten und ihre Prozesse an die Community anpassen müssen. Praktiken, die in der internen Zusammenarbeit sowie in partnerschaftlichen Kollaborationen mit anderen Firmen erprobt sind, funktionieren nicht zwingend in einer sich selbst organisierenden Gemeinschaft. Auch die Open-Source-Communities mussten teilweise ihre Prozesse anpassen, um möglichst optimal mit den Firmen zusammenarbeiten zu können. Inzwischen haben beide, sowohl die Communities als auch die Firmen, Strategien erarbeitet, um ihre eigenen Interessen nicht durch andere zu korrumpieren. Dazu werden – als für beide Seiten nützliche Maßnahme – unabhängige Non-Profit-Organisa-

tionen gegründet, die eine Vermittlerrolle einnehmen. Als erstes Projekt lancierte dies der Apache-Webserver mit der Apache Software Foundation. Eine treibende Kraft war dabei IBM:

„Als IBM im Jahre 1998 mit den Apache-Leuten einen Vertrag abschließen wollte, um den Webserver in ihr Produkt Websphere zu integrieren, waren die Konzernanwälte mehr als verblüfft. Es gab keine Organisation als Vertragspartner. Auf die Frage, wie denn nun Apache organisiert sei, kam die Antwort, es sei eine Website. ‚Habe ich das richtig verstanden, wir machen hier einen Vertrag mit einer Website?‘“ (Wolf 2002, S. 48)

Die Non-Profit-Organisation nimmt einerseits die Rolle der juristischen Person ein, andererseits ist sie für die Infrastruktur verantwortlich. Wichtig dabei ist, dass sie ihre Unabhängigkeit gegenüber der Firma wahren kann. Bei der Apache Software Foundation wurde dies beispielsweise so gelöst, dass nur Personen und keine Organisationen Mitglieder sein können und dabei grundsätzlich jede Person dieselben Rechte hat, unabhängig von ihrem finanziellen Beitrag. Zusätzlich werden die Ausschüsse von den Mitgliedern demokratisch gewählt, wobei wiederum gilt, dass jede Person eine Stimme hat (Fielding 1999).⁸⁹

Dass die rechtliche Unabhängigkeit eines Projektes für die Beteiligung von kommerziellen Organisationen elementar ist, hat unter anderem das Eclipse-Projekt gezeigt. Die ursprünglich von IBM intern für den Eigenbedarf bereitgestellte Entwicklungsplattform wurde schon bald geöffnet, um sie insbesondere für Partnerfirmen interessant zu machen (O'Mahony et al. 2005). Erst die Loslösung des Projekts von der IBM durch die Übergabe der Rechte an die unabhängige Eclipse Foundation konnte andere Firmen davon überzeugen, sich zu beteiligen (Spaeth et al. 2008). Heute ist Eclipse der de-facto-Standard in Bereich der Java-Entwicklungsplattformen und hat einen Marktanteil von weit über 50 %.

3.3.4 Zwischenfazit

FLOSS wird nicht als Modell gesehen, das für die gesamte Softwareindustrie Gültigkeit hat,

⁸⁹ Details dazu sind auch auf der Apache-Homepage unter <http://www.apache.org/foundation/how-it-works.html> [17.08.2009] nachzulesen.

wie es Richard Stallman und die Free Software Foundation fordern. Dessen Berechtigung wird nur in Teilbereichen gesehen, wo sich ein erfolgreiches Geschäftsmodell etablieren kann. Allerdings kann mit der primären Wertschöpfung nur bedingt eine genügend hohe Rendite erreicht werden. Mehr Erfolg verspricht ein Geschäftsmodell, das auf der sekundären Wertschöpfung aufbaut. Erste Ansätze einer strategischen Betrachtung sind bereits sichtbar. Diese zeigen Auswirkungen der Mitarbeit von Firmen an Open-Source-Projekten auf die Community sowie die Unternehmen, wie z. B. die Gründung der unabhängigen Non-Profit-Organisationen. Die Diskussion konzentriert sich jedoch weitgehend auf die pragmatische Frage, wie mit FLOSS direkt Geld zu verdienen ist. Eine Betrachtung aus einer gesamtheitlichen, strategischen Unternehmenssicht wird noch zu leisten sein.

3.4 Die dritte Phase: Strategie

„Open Source: The Model Is Broken“ titelte provokativ ein Artikel in der BusinessWeek (Cohen 2008). Im Artikel wird weiter ausgeführt, dass das auf Support und Service basierende Geschäftsmodell langfristig nicht funktionieren könne, weil Open-Source-Software schlicht zu gut dafür sei. Der Autor sieht die Zukunft von Open Source vielmehr darin, dass sich in Firmen die Erkenntnis einer kollaborativen und damit viel kostengünstigeren Entwicklung einer Software-Infrastruktur zunehmend etabliert. Dafür ist allerdings eine strategische Ausrichtung erforderlich. IBM erkannte dies als eine der ersten Firmen und hat sich bereits Ende der 1990er Jahre zu FLOSS bekannt. Andere Unternehmen, wie etwa Sun, realisierten die Chancen erst später und rüsten nun massiv nach. Microsoft schließlich, das in seinem Kerngeschäft von Open-Source-Software bedroht wird wie kaum ein anderes großes Unternehmen, nähert sich mittlerweile schrittweise an. Werner (2007) erläutert anhand der im Auswärtigen Amt gemachten Anwendererfahrungen mit Linux, dass sich entscheidende Vorteile von FLOSS wie Unabhängigkeit und Anpassbarkeit nur durch eine strategische Integration in den Produktionsprozess erreichen lassen.

Wie der „FLOSS Final Report“ – eine von der EU in Auftrag gegebene Studie von 2002 – aufzeigt, engagiert sich rund ein Drittel der 25 größten Softwarefirmen in erheblichem Maße und in strategischer Weise in der Entwicklung von Open-Source-Software, insbesondere von Linux (Wichmann 2002). Heute dürfte die Zahl deutlich höher sein, engagieren sich doch in-

zwischen auch in dieser Liste aus dem Jahre 2002 noch nicht enthaltene Firmen wie Oracle und Microsoft – wenn auch nicht beim direkten Konkurrenten Linux – in Open Source.

Ergebnisse der aktuelleren, ebenfalls von der EU in Auftrag gegebene MERIT-Studie zum ökonomischen Einfluss von Open-Source-Software auf den europäischen IKT-Sektor zeigen hingegen, dass lediglich 20 % des Codes von Firmen beigetragen werden (Ghosh 2006). Gemäß dieser Studie sind dabei die zehn engagiertesten Firmen (in der Reihenfolge der Anzahl Codebeiträge) Sun, IBM, Red Hat, Silicon Graphics, SAP, MySQL, Netscape, Ximian, Realnetworks und AT&T. Diese Zahlen beruhen auf dem Software-Verzeichnis der Debian-Linux-Distribution, einem traditionell „privaten“ Projekt, weshalb die Zahlen eher als zu niedrig zu interpretieren sind. Wie die Auswertungen z. B. des Linux-Kernels zeigen, liegt dort die Firmenbeteiligung bei ungefähr 70 % (Kroah-Hartman et al. 2008). Die beiden Werte lassen sich auch dahin gehend interpretieren, dass vor allem Software, die als Infrastruktur bezeichnet werden kann, von Firmen aktiv unterstützt wird. Bei Anwendungssoftware hingegen, insbesondere bei Software für den Konsumentenmarkt, bietet sich weniger offensichtlich ein Geschäftsmodell an.

Open Source bedeutet jedoch mehr, als gute Software in einem kollaborativen Entwicklungsprozess zu erstellen. Es muss vielmehr im Zusammenhang mit einer generellen Öffnung der in einer globalisierten Welt zu eng gewordenen Firmengrenzen für IT-Unternehmen gesehen werden.

3.4.1 „Openness“: Ein erweitertes Verständnis

Die Erkenntnis ist deutlich gewachsen, dass ein isoliertes Vorgehen im Zeitalter des Internets, d. h. des raschen Zugriffs auf weltweite Informationen, nicht mehr angebracht ist. Open-Source-Software allein führt jedoch noch nicht zum Ziel der vollständigen Interoperabilität⁹⁰. Zwar wird durch die Offenlegung des Quelltextes Herstellerunabhängigkeit erreicht. Dies trifft für die Produktunabhängigkeit und somit für die volle Flexibilität jedoch noch nicht zu⁹¹.

90 Unter Interoperabilität versteht man, dass zwei oder mehr Systeme, Techniken oder Organisationen zusammenarbeiten können.

91 Suhr (2008) führt in dem von ihm definierten Benchmark zur Messung der Offenheit von IT-Artefakten insgesamt fünf Kriterien auf: Unabhängigkeit, Kompatibilität, Interoperabilität, Verfügbarkeit und die fehlende Betroffenheit durch Patente.

Erst durch Innovationen in der Zusammenarbeit wird dies möglich sein. Dies skizziert Jollans (2006) anhand des Begriff „Community Innovation“ bzw. des Konzept „Open Computing“ (siehe Abb. 6). Damit meint er das Zusammenspiel der drei Komponenten „Open Architecture“, „Open Standards“ und „Open Source“, bei dem eine vollständige Interoperabilität erreicht werden kann. Das Ziel von „Open Computing“ ist die Flexibilität einer modularen Integration von Funktionalität sowie die Unabhängigkeit von Herstellern, sowohl bei der Hardware als auch bei der Software.

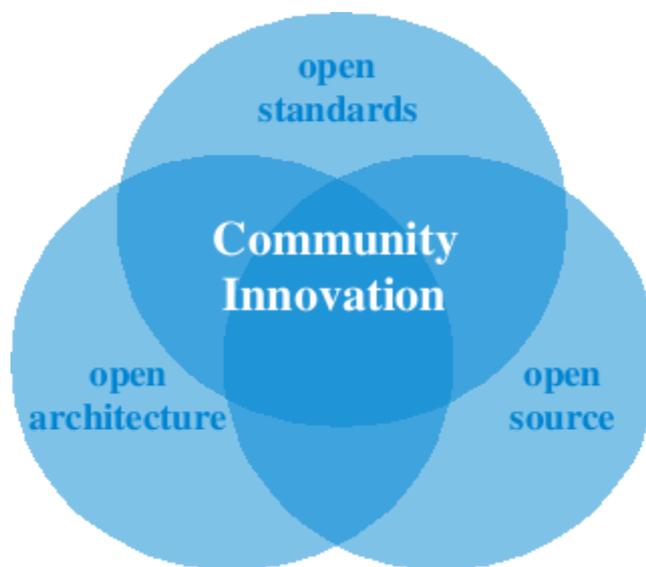


Abbildung 6: Open Computing
(nach Jollans 2006, S. 3)

Roy Fielding (2008), einer der Protagonisten beim Webserver Apache, fasst unter dem Begriff „Open Architecture“ – siehe dazu auch die wegweisende Dissertation von Oreizy (2000) – Software zusammen, die sich in erster Linie durch offene Entwicklung und offene Schnittstellen auszeichnet und so erst ein Ökosystem von Anbietern erlaubt. Fielding fasst erfolgreiche (Open-Source-)Software mit dem Ansatz der „Open Architecture“ wie folgt zusammen:

„a software architecture designed to promote anarchic collaboration through extensions while preserving control over the core interface “ (Fielding 2008, S. 17)

Als Beispiele nennt er unter anderem Emacs, den Apache Webserver, Mozilla Firefox sowie den Linux-Kernel. Dieser Aufzählung hinzuzufügen ist auch das Eclipse-Projekt (O'Mahony

et al. 2005). Dabei handelt es sich um Software, die man als Infrastruktur bezeichnen kann. Auch Varian und Shapiro (2007) schließen sich der Ansicht von Fielding an. Ihrer Meinung nach lässt sich das Ziel der Interoperabilität nur durch offene Standards erreichen. Zwar müssen Standards an sich offen zur Verfügung stehen, damit sie überhaupt funktionieren können. Eine offene Entwicklung sowie eine offene Anpassbarkeit sind jedoch nicht zwingend notwendig. Nur Standards, die alle diese drei Formen der Offenheit bieten und zudem nicht patentbehaltet sind, werden jedoch als offene Standards bezeichnet und z. B. von der EU unterstützt. Varian und Shapiro sehen zudem durch die offenen Standards eine nachhaltige Veränderung in der Strategie von IKT-Firmen:

„Offene Standards und Interoperabilität führen dazu, den Konkurrenzschwerpunkt in der Computerindustrie vom Wettbewerb um den Standard auf den Wettbewerb innerhalb des Standards zu verlagern.“ (Varian und Shapiro 2007, S. 127)

Der von Chesbrough (2003) eingeführte Begriff der „Open Innovation“ versteht unter Offenheit grundsätzlich etwas anderes. Geöffnet werden sollen die Firmengrenzen dahin gehend, dass einerseits Ideen von außerhalb aufgenommen werden und andererseits Innovationen, die selbst nicht vermarktet werden, anderen (kostenpflichtig) zur Verfügung gestellt werden können. Insofern ist der von Jollans verwendete Begriff „Community Innovation“ als Synonym zu „Open Innovation“ zu verstehen.

Interoperabilität ist nicht nur das Ziel des „Open Computing“, sondern auch Voraussetzung für das Funktionieren der „Community Innovation“. Diese Interoperabilität basiert wiederum auf einer stabil funktionierenden Infrastruktur, die somit zunehmend von strategischer Bedeutung wird.

3.4.2 Der Infrastrukturbegriff

Eine stabil funktionierende Infrastruktur ist für jede Gesellschaft und die in ihr agierenden Individuen und Organisationen elementar. Es ist deshalb aus strategischer Sicht wichtig, in diesem Bereich neben der Stabilität auch eine möglichst große Unabhängigkeit von Entscheidungen Dritter zu erreichen. Diese Unabhängigkeit kann insbesondere dadurch erreicht werden, dass die Infrastruktur in die Allmende einfließt.

In der (digitalen) Allmende befindet sich auch Freie und Open-Source-Software. Dass der Erfolg von FLOSS vorwiegend im Infrastrukturbereich gesehen wird, ist unter diesem Aspekt nachvollziehbar. Dabei wird unter Infrastruktur im Softwarebereich diejenige Software verstanden, die auf einem Server läuft. Linux hat sich denn auch im Serverbereich einen beträchtlichen Marktanteil sichern können, auf dem Desktop bisher jedoch lediglich eine zu vernachlässigende Rolle gespielt.⁹²

In der IT-Fachwelt wird der Infrastrukturbegriff etwas weiter gefasst. Darunter ist eine technologische Infrastruktur zu verstehen, die Organisationen bereithalten, „damit überhaupt gehandelt werden kann“ (Lutterbeck 2007, S. 3). Neben der Hardware und dem Betriebssystem zählen auch Applikationen für E-Mails, Bürokommunikation, Datendienste etc. dazu, die sich sowohl auf dem Server als auch auf dem Client befinden können (z. B. Mailserver und Mailclient).

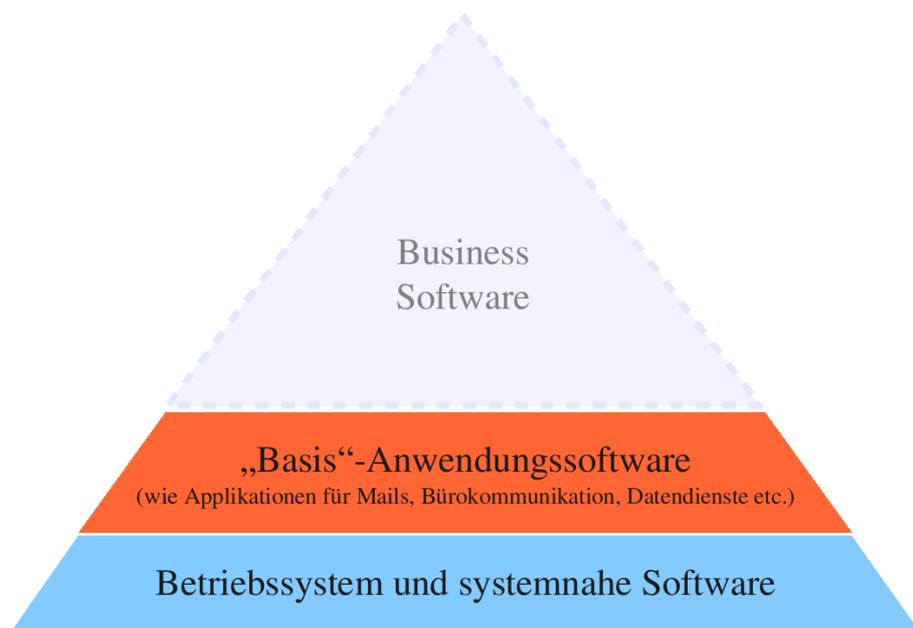


Abbildung 7: Software-Infrastruktur

Die Abbildung 7 unterscheidet nicht zwischen Software, die auf dem Server, und Software, die auf dem Client läuft, sondern orientiert sich an der im vorherigen Absatz zitierten Defini-

⁹² Die MERIT-Studie (Ghosh 2006) geht z. B. von einem Linux-Marktanteil auf Servern von über 50 % aus. Sie bezieht sich dabei auf eine Quelle aus dem Jahre 2005. Auf dem Desktop hingegen befindet sich Linux derzeit auf der 1%-Schwelle (<http://marketshare.hitslink.com/os-market-share.aspx?qprid=9> [17.08.2009]).

tion von Lutterbeck. Betriebssystem und systemnahe Software können als Infrastruktur im engeren Sinne, die „Basis“-Anwendungssoftware als Infrastruktur im weiteren Sinne bezeichnet werden. Beide dienen jedoch noch nicht der eigentlichen Geschäftstätigkeit, die stark von der jeweiligen Art des Betriebes abhängt.

Unabhängigkeit im Infrastruktur-Bereich kann nicht alleine durch den Konsum von FLOSS erreicht werden. Wie das Beispiel der vertikalen Integration im Deutschen Auswärtigen Amt zeigt (Auener 2008), ist die durch die Freie Software gewährte Unabhängigkeit nur realisierbar, wenn man sich auch an den jeweiligen Projekten beteiligt. Diese Erkenntnis setzt sich im Sinne eines Reifeprozesses allmählich durch und zeigt sich zunehmend in Strategien einer wachsenden Anzahl von Unternehmen in- und außerhalb der IKT-Branche.

3.4.3 Ein Reifeprozess-Modell

Weder im Privatbereich noch im kommerziellen Umfeld ist es üblich, ausschließlich freie und quelloffene Software zu nutzen und zu unterstützen. Vielmehr soll ein (Lern-)Prozess angestoßen und verfolgt werden, in dem zu Beginn eher der kurzfristige Nutzen durch die Nutzung von Open-Source-Software steht. Mit den positiven Erfahrungen und den wachsenden Bedürfnissen entsteht eine nahezu natürliche Entwicklung, sich als Firma oder Privatperson zunehmend zu beteiligen. Dabei können vor allem Firmen ihre eigene Strategie aufbauen und die eigenen Prozesse anpassen.⁹³

Carbone (2006; 2007) hat über ein halbes Jahrzehnt hinweg erforscht, wie sich IT-Firmen hin zur FLOSS-Bewegung orientieren. Daraus ist ein bisher weitgehend unbeachtetes, umfassendes Konzept entstanden, das sich in kommerziellen Firmen strategisch nutzen lässt. Die in Abbildung 8 aufgezeichneten Schritte des Reifeprozess-Modells sind im Folgenden kurz erläutert.

⁹³ Eine Übersicht zu den Strategien bietet z. B. West (2008).

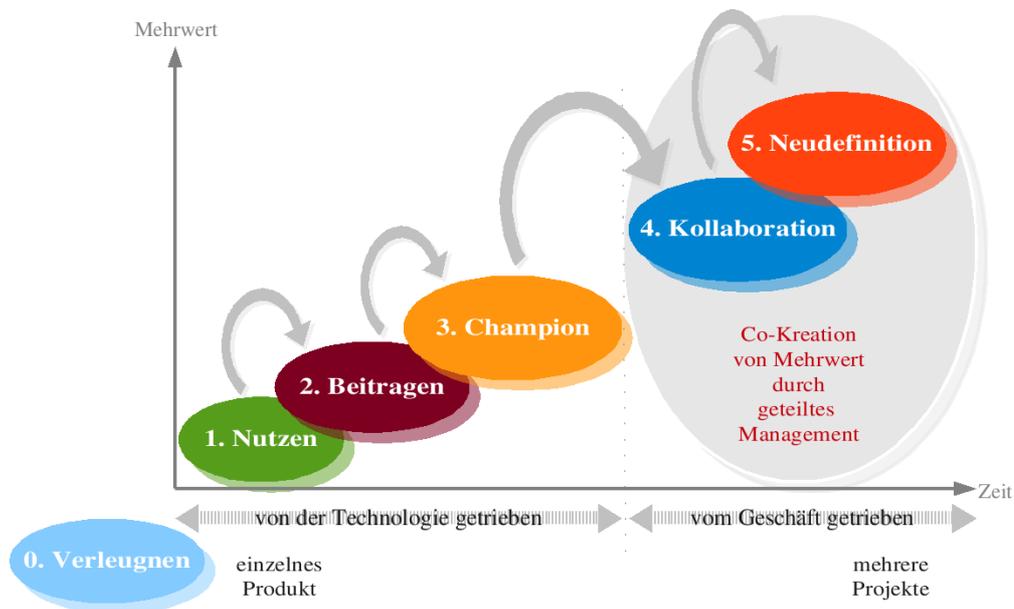


Abbildung 8: Reifeprozess-Modell von Firmen in der Open-Source-Beteiligung
(nach Carbone 2007, S. 5)

Nutzen: Die Stufe des Nutzens basiert einerseits auf der Motivation, Kosten zu senken, andererseits auch darauf, die Phase „time to market“ zu verkürzen. Geführt wird der Open-Source-Einsatz von technologisch orientierten Personen, die oft keine Managementfunktion haben. Die Nutzung ist meist noch unkoordiniert und weitgehend unkontrolliert. Die Firmen beginnen jedoch oft damit, Open Source nicht nur anzuwenden, sondern auch voranzutreiben. Der Nutzen für die Open-Source-Community ist in dieser Phase nahezu inexistent.

Beitragen: Die Motivation, zum Beitragenden zu werden, liegt darin, fehlende, aber selbst benötigte Funktionalität hinzuzufügen sowie die Softwarequalität allgemein zu verbessern. Die Firma passt ihr Verhalten in ersten Geschäftsbereichen an. Und die Führung wird von Produktmanagern oder Technologieverantwortlichen übernommen, die konkrete Codebeiträge bestimmen und diese auch koordinieren. Die Beiträge können darin bestehen, Zeit zur Verfügung zu stellen, geschriebenen Code herzustellen oder Geld zu spenden. Für die Open-Source-Community liegt der Nutzen in mehr Funktionalität und einer besseren Softwarequalität.

Champion: Erst mit dem Erreichen der dritten Stufe beginnt die strategische Auseinandersetzung mit Open-Source-Software. Geschäftsmodelle beginnen hier zu greifen. Zum Champion wird, wer auf FLOSS ein Geschäftsmodell aufbaut und sich ein entsprechendes Know-how

erarbeitet. Champions fügen gezielt neue Funktionalität hinzu und fördern so eine evolutionäre Entwicklung des unterstützten Projektes. In dieser Stufe beginnt Open-Source-Software zur „Chefsache“ zu werden und erreicht die höheren Managementebenen. Neben konkreten Codebeiträgen wird auch gezielt im inneren Kern des Projektes mitgearbeitet, indem die Firma entsprechende Personalressourcen zur Verfügung stellt. Diese werden oft aus der Community angeworben. Die Firma versucht gezielt, Kontrolle über das Projekt zu gewinnen, zumindest wird es intensiv beobachtet. Die Koordination beschränkt sich nicht nur auf die eigenen Angestellten, sondern über geeignete Maßnahmen (wie etwa gesponserte Aktivitäten) werden auch externe Community-Mitglieder einbezogen. Zudem wird die Community beim Bekanntmachen des Projektes unterstützt. Durch einen Champion wird ein Projekt stark vorangetrieben, und durch dessen Engagement ist eine längerfristige Stabilität des Projekts gewährleistet.

Kollaboration: Durch die aktive Kollaboration versucht eine Firma gezielt, eine führende Position in der Community zu erreichen. Dies führt zu einem Marktvorteil für ein eigenes Produkt bzw. im eigenen Marktsegment. Die Führung wird vom strategischen und operativen Verkaufsmanagement übernommen. Die geleisteten Beiträge werden somit einerseits von der längerfristigen Strategie, andererseits von den aktuellen Kunden- und Marktanforderungen bestimmt und werden zunehmend lösungs- anstatt „nur“ produktorientiert. Die kollaborierende Firma versucht gezielt, den Markt und ihre Kunden in eine Open-Source-Richtung zu steuern, um auch Redundanzen in der Software-Entwicklung innerhalb des ganzen IT-Segments zu eliminieren. Sie investiert in Open Source und bekennt sich auch klar dazu. Die Open-Source-Community erhält durch die Kollaboration ein größeres Gewicht und die so entwickelte Software kann gut zu einem Industriestandard werden. Der Entwicklungsprozess wird zudem beschleunigt.

Neudefinition: Am Ende des FLOSS-Reifeprozesses steht eine Neudefinition des eigenen Geschäfts, in deren Zentrum eine Änderung des Mehrwerts der Firma steht, den sie ihren Kunden offeriert. Vorangetrieben wird der Prozess vom Top-Management sowie den Technologie-Verantwortlichen (CTO). Neben den konkreten Codebeiträgen zu den für das eigene Geschäft relevanten Projekten wird auch in Werkzeuge investiert, die das Arbeiten in und mit Open Source vereinfachen. Mithilfe eines neuen Partnernetzwerks und neuer Geschäftsmodelle werden den Kunden zunehmend Lösungen auf Open-Source-Basis angeboten. Die Community gewinnt dadurch einen großen Pool an Beitragenden, und aufgrund des neuen Partner-

netzwerkes entstehen neue Optionen in der Zusammenarbeit mit anderen Projekten sowie in der Rekrutierung neuer Community-Mitglieder.

Zusammenfassend betont Carbone, dass der von ihm beschriebene Reifeprozess weder eine rein technologische noch eine isoliert unternehmensstrategische Entwicklung ist, sondern gleichzeitig auf den Ebenen Technologie, Geschäft und Markt Auswirkungen zeigt.

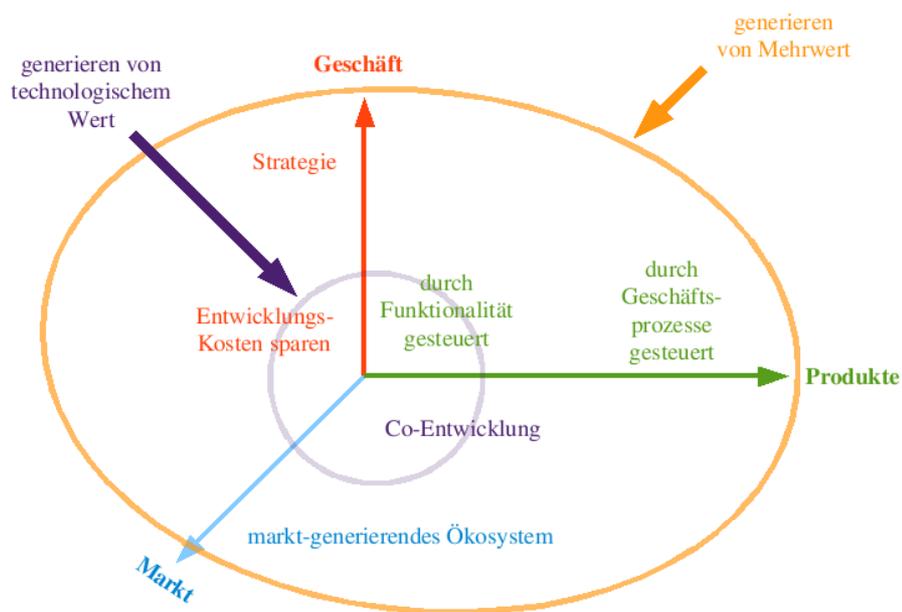


Abbildung 9: Betriebs- und volkswirtschaftliche Auswirkungen des Reifeprozess-Modells (nach Carbone 2007, S. 10)

In der Abbildung 9 wird deutlich, dass der durch ein einzelnes Unternehmen generierte Mehrwert aufgrund der strategischen Unterstützung von FLOSS zunimmt und sich über das Ökosystem und die freie Lizenz der Software auch auf den Markt im Gesamten ausweitet. Ein Unternehmen, das FLOSS aus strategischer Sicht unterstützt, ist somit ein Katalysator in seinem Segment. Der Prozess, der bereits unter Bezugnahme auf Schumpeter (1961) in der Ausgangslage (Kapitel 1.1) beschrieben wurde, ist folglich nicht nur aus unternehmerischer, sondern auch aus Sicht einer Volkswirtschaft interessant, da er globales Wachstum generieren kann. Dies wird durch zahlreiche andere Studien bestätigt (siehe z. B. Ghosh 2006). Allerdings gibt es bislang noch keine konkrete Klassifizierung von Firmen in diesem Modell. In einer persönlichen E-Mail an den Verfasser hat Carbone seine aktuellen, noch nicht publizierten Datenanalysen zumindest dahin gehend erklärt, dass aufgrund der heutigen Situation Anzeichen dafür bestehen, IBM hätte die Stufe der Neudefinition erreicht (Carbone,

persönliche Kommunikation 2008).

3.4.4 Mitarbeit in kommerziellen Firmen

Firmen, die ihre Strategie partiell oder vollständig auf FLOSS ausrichten, sehen sich insbesondere zwei Herausforderungen in der alltäglichen Zusammenarbeit gegenüber. Zum einen gilt es, die eigene Mitarbeit an die Regeln der Community anzupassen. Zum anderen muss die Firma weiterhin darauf bedacht sein, dass sie ihre Interessen wahrnimmt. Dahlander und Magnusson (2006) haben anhand von vier Fallstudien mit skandinavischen Open-Source-Firmen festgestellt, dass es für Projekte, die stark von Firmen dominiert sind, schwierig ist, Freiwillige zu rekrutieren und zu halten. Offensichtlich fühlen sie sich durch die Firmenbeteiligung irritiert. Die Autoren schließen daraus, dass nur ein symbiotisches Verhalten der Firmen gegenüber der Community zu einem nachhaltigen Geschäftsmodell führt.

Henkel (2009) sieht in den eher seltenen Fällen, in denen sich ein angestellter FLOSS-Entwickler mit der Ideologie der Freie-Software-Bewegung identifiziert, einen Prinzipal-Agenten-Konflikt⁹⁴. Ansonsten verhalten sich die Entwickler – seinen umfassenden empirischen Daten gemäß – durchaus im Interesse ihres Arbeitgebers. Dem widersprechen zwei Studien (Dahlander und Magnusson 2006; Rossi und Bonaccorsi 2006) und konstatieren, dass das Gedankengut aus der Open-Source-Bewegung durchaus in die Unternehmen fließt:

We can reasonably conclude that the social dimension of the Open Source movement has a chance to survive to its evolution into an economic reality through individuals that transfer hacker culture to Open Source companies. (Rossi und Bonaccorsi 2006, S. 103)

Dahlander und Magnusson bewerten diesen Transfer nur als bedingt positiv, da die Unternehmen dieses für eine Firma fremde Gedankengut in der Regel nicht als solches erkennen und deshalb in ihren Entscheidungen fehlgeleitet werden können:

94 Die Prinzipal-Agenten-Theorie ist Bestandteil der Neuen Institutionenökonomik (North 1990; 1992). Sie geht davon aus, dass in einer Vertragssituation – u. a. in einem Anstellungsverhältnis – jeder Akteur seinen individuellen Nutzen optimieren möchte. Dies wird jedoch dadurch erschwert, dass nur beschränkt Informationen über die Ziele und das geplante Handeln des Gegenübers vorhanden sind. Diese „Asymmetrie der Informationen“ kann sowohl der Prinzipal (Auftraggeber) als auch der Agent (Auftragnehmer) ausnutzen, was zu Konflikten führen kann.

However, in the case of OSS firms, we can see that the norms and values of the open source movement are not only diffused in OSS communities, but often also influence management in OSS firms, and there is thus a potential risk that the social capital in this specific setting potentially biases decisions and limits the actions of these specific firms. This points to the necessity for OSS firms to consider their social capital as a resource that can be both positive and negative, and consequently the need to deliberately manage these structures. (Dahlander und Magnusson 2006, S. 115)

Sie listen deshalb sieben Management-Herausforderungen auf, die sich für die Firmen aufgrund ihrer Zusammenarbeit mit einer FLOSS-Community ergeben:

- Respektierung der Normen und Werte der Community
- Förderliche Verwendung der Lizenzen
- Attraktive Rekrutierungsstrategien für Entwickler und Nutzer
- Verwaltung des Ressourcenverbrauchs in Bezug auf die Community-Arbeit
- Zusammenführung von Freiwilligen- und entlohnter Arbeit
- Lösung von Spannungen in Bezug auf Kontrolle und Besitzanspruch
- Akzeptanz für die Verwendung von FLOSS in einem kommerziellen Geschäftsmodell sowie die Vermeidung von direkten Konflikten

Für ein nachhaltig erfolgreiches Geschäftsmodell mit FLOSS kann es sich eine Firma kaum leisten, vollständig von den Entscheidungen und dem Goodwill einer unabhängigen Gemeinschaft abhängig zu sein.⁹⁵ Sie wird deshalb versuchen, ihren Einfluss auf die jeweiligen Projekte zu erhöhen. Dies kann sie einerseits sehr direkt erreichen, indem sie Code beisteuert. Da die Communities in der Regel meritokratisch funktionieren, erreicht sie dies, indem eine Firma Softwareentwickler vorrangig an einem Projekt arbeiten lässt. Leisten diese Personen qualitativ und quantitativ gute Arbeit, so gewinnen sie zudem eine Reputation in der Community (Raymond 1999; Lerner und Tirole 2002), die auch auf die Firma übergeht. Ziel ist es dabei, zentrale Positionen zu besetzen, um zumindest teilweise Ressourcen managen zu können, ohne im juristischen Sinne Besitzansprüche zu haben:

⁹⁵ Ein wichtiger Grund zur Unterstützung von FLOSS ist ja bei vielen Firmen auch die Eliminierung von Abhängigkeiten gegenüber anderen Firmen, wie etwa gegenüber Microsoft.

„Access to communities allows firms to get access to resources that cannot be bought in the market.“ (Dahlander und Wallin 2006, S. 1247)

Die im Kapitel 3.2.1 beschriebene Hierarchie innerhalb von FLOSS-Projekten, bestehend aus der Kerngruppe, den Beitragenden sowie den Benutzern, bekommt aus Sicht der Firmen neue Facetten. Wie Dahlander und Wallin in ihrem Artikel „A Man on the Inside: Unlocking communities as complementary assets“ (2006) belegen können, haben bezahlte FLOSS-Entwickler tendenziell ein größeres Netzwerk. Sie argumentieren, dass es sich um eine Strategie der Firmen handelt, die aufgrund der fehlenden Besitzansprüche verlorene Kontrolle auf diesem Weg zumindest teilweise wiederzuerlangen. Während es in kleineren FLOSS-Communities noch relativ einfach ist, diesen zentralen Status zu erhalten, ist dies bei größeren Projekten wie dem Linux-Kernel nicht der Fall, sondern es bedarf meist mehrerer Jahre der engagierten Mitarbeit. Es ist deshalb bei großen IKT-Firmen auch durchaus gebräuchlich, diesen „man on the inside“ aus der Community „einzukaufen“. Diese Praxis haben z. B. Novell/Suse mit Greg Kroah-Hartman und Red Hat mit Alan Cox praktiziert. Dass es daneben auch ein wesentlicher Erfolgsfaktor für den Einsatz von FLOSS ist, das entsprechende Know-how intern aufzubauen, zeigt das Beispiel der vertikalen Integration des Auswärtigen Amtes in Deutschland (Auener 2008).

Dahlander und Magnusson (2006) erachten es zudem als plausibel, dass sich freiwillige viel mehr als angestellte Softwareentwickler an intrinsischen und sozialen Motiven orientieren. Diese These wird durch Luthiger Stoll (2006) unterstützt, der empirisch belegen konnte, dass Softwareentwicklung im Community-Umfeld mehr Spaß macht als in einem kommerziellen Umfeld.

3.4.5 Zwischenfazit

Freie und Open-Source-Software ist mittlerweile Mainstream. Es gibt kaum eine namhafte IKT-Firma, die keine Open-Source-Software einsetzt bzw. sich nicht an Open-Source-Projekten beteiligt. Jedoch ist die Anzahl derjenigen Firmen noch in der Minderheit, die in Bezug auf FLOSS eine proaktive Strategie verfolgen (Laisné 2008). Die meisten Institutionen dürften sich in einem „FLOSS-Reifeprozess“ befinden. Sie sehen die Berechtigung freier Software im gesamten Softwareportfolio und sind zunehmend weniger opportunistisch in ihrer dies-

bezüglichen Wahl. Sie nutzen und fördern FLOSS aus strategischen und weniger aus kosten-senkenden Gründen. Open-Source-Software wird zudem strategisch in den größeren Zusammenhang des Konzepts des „Open Computing“ gesetzt.

Die bisherige fachliche und wissenschaftliche Diskussion konzentriert sich dabei jedoch stark auf die relativ jungen OSS-Firmen und vernachlässigt dabei die Besonderheiten der etablierten IKT-Firmen. Insbesondere die Rolle der in einem Angestelltenverhältnis wirkenden Open-Source-Entwickler wird, wenn überhaupt, nur aus der Firmensicht betrachtet. Die Sicht der Individuen wurde bislang hingegen kaum beleuchtet.

3.5 Konklusion zum Stand der Diskussion

Die Diskussion zu Open-Source-Software hat sich in einer Frühphase vor allem mit den spezifischen Fragen der Freiwilligkeit innerhalb der Community befasst. Neben der Frage der Motivation der Beitragenden stand dabei auch das spezifische Entwicklungsmodell, vor allem in Bezug auf die selbst organisierende Koordination, im Zentrum. Mit der zunehmenden Verbreitung der Software hat sich der Fokus auf die Wirtschaftlichkeit sowohl auf Seiten der Nachfrage als auch des Angebots verlegt. Auf der Nachfrageseite versuchte man zu belegen, dass die Gesamtkosten von Open Source gegenüber proprietärer Software niedriger sind. Auf der Angebotsseite wurden Möglichkeiten, Nutzen und Risiken der Open-Source-Geschäftsmodelle erforscht. In jüngerer Zeit wird zunehmend auch im strategischen Bereich geforscht, einerseits aus volkswirtschaftlicher, andererseits aus unternehmerischer Sicht. Vor allem die EU fördert Open-Source-Software aus Gründen der Unabhängigkeit der Regierungen gegenüber (vornehmlich amerikanischen) Konzernen.

Die drei zentralen Begriffe „Idealismus“, „Pragmatismus“ bzw. „Strategie“ wurden bis hierhin unabhängig voneinander präsentiert und diskutiert, d. h., Wechselwirkungen zwischen den drei Phasen sind kaum berücksichtigt. Diese Beschränkung führt dazu, dass das „Phänomen Open-Source-Software“ nicht in seiner ganzen Tragweite erfasst werden kann bzw. die Argumentation eindimensional bleibt.

Die wechselseitigen Beziehungen lassen sich schematisch wie folgt darstellen:

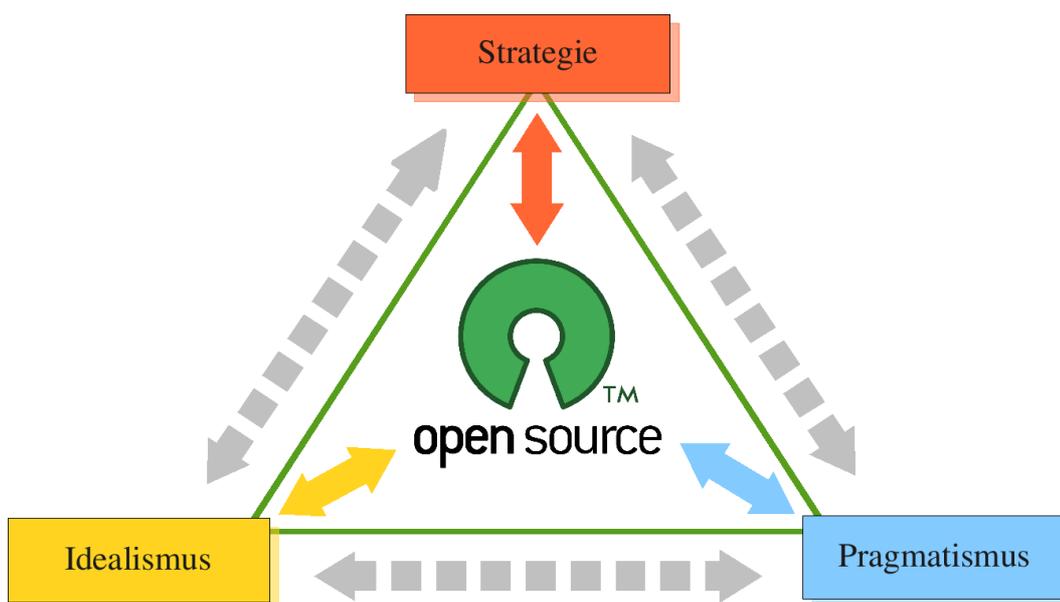


Abbildung 10: Wechselwirkungen im Open-Source-Ökosystem

Während die farbigen Bereiche in der Abbildung 10 bereits in den drei Phasen beschrieben wurden, sind die grau dargestellten Wechselwirkungen in der Diskussion bisher weitgehend vernachlässigt worden.

Angesichts eines generellen Mangels an empirischen Daten zu Open-Source-Software – insbesondere im unternehmerischen Umfeld – beschränken sich die wenigen Studien, die sich den Wechselwirkungen gewidmet haben, auf kleinere, sogenannte Open-Source-Companys. Empirische Daten zu global agierenden Konzernen fehlen fast vollständig. Zusätzlich fand die Diskussion dabei weitgehend aus Sicht der Firmen statt. Forschung über die Angestellten, die sich im Spannungsfeld zwischen den pragmatischen und strategischen Interessen ihres Arbeitgebers und den eher ideellen Werten der Community befinden, gibt es wiederum nur für kleine Firmen, aber nicht für die großen Konzerne. Diese stellen allerdings die Mehrzahl der Entwickler zur Verfügung.

Basierend auf dieser Konklusion soll in dieser Dissertation ein Beitrag dazu geleistet werden, dass einerseits empirische Ergebnisse generiert werden – diesbezüglich gibt es bis dato ein Ungleichgewicht zugunsten theoretischer Arbeiten. Andererseits sollen die Wechselwirkungen mit einem Fokus auf die Softwareentwickler von IKT-Konzernen untersucht werden.

3.6 Forschungsfragen

Aufgrund der in der Konklusion beschriebenen Forschungslücke liegt der Fokus in dieser Studie auf der unternehmerischen Sicht der Open-Source-Softwareentwicklung, insbesondere in großen, etablierten IT-Unternehmen. Dieser Fokus basiert auf der Prämisse, dass gerade diese Firmen die Open-Source-Software entscheidend vorwärtstreiben.

Wie u. a. die Forschungsstudien von Rossi und Bonaccorsi (2006) sowie Dahlander und Magnusson (2006) anhand von kleinen „Open-Source-Firmen“ aufzeigen, löst die Mitarbeit an freien Softwareprojekten in einem kommerziell orientierten Umfeld verschiedene betriebliche Prozesse und mitunter auch Unstimmigkeiten aus. Interessant wird sein, wie sich dies in großen Firmen auswirkt. Hier setzt die vorliegende Studie mit den folgenden Forschungsfragen an:

1. Wie gestaltet sich die Arbeit von Open-Source-Entwicklern innerhalb großer IKT-Firmen? Welche Unterschiede zur „Closed-Source“-Entwicklung gibt es?
2. Welche Spannungsfelder ergeben sich zwischen der Open-Source-Community und großen IKT-Firmen?
3. In welchem Ausmaß beteiligen sich große IKT-Firmen an der Open-Source-Softwareentwicklung, und wie lässt sich dieser Beitrag quantifizieren?

Aufgrund der Tatsache, dass dieses Feld bisher weitgehend unerforscht ist, war es die Absicht, das Thema im Sinne der Grounded Theory (Glaser und Strauss 1967) möglichst offen und unvoreingenommen anzugehen. Es war deshalb wichtig, sich nicht durch eine eng gefasste Fragestellung einzuschränken und mögliche Erkenntnisse im Feld dadurch auszuschließen (Eisenhardt 1989).

Während der Datenerhebung und der iterativ geführten Datenanalyse hat sich in der Folge eine wichtige neue Frage herauskristallisiert:

4. Anhand welcher Dimensionen lassen sich Open-Source-Entwickler in großen IKT-Firmen charakterisieren und welche Idealtypen ergeben sich daraus?

Teil II: Methodisches Konzept

4 Methodische Hinführung

4.1 Fallstudien-Design

Da zu der Fragestellung dieser Studie bislang noch keine empirischen Daten vorliegen, ist ein exploratives Vorgehen angemessen. Im Bereich von Open-Source-Software in großen IKT-Firmen zu forschen heißt auch, dass nur eine beschränkte Anzahl von Firmen für die Datenerhebung zur Verfügung steht. Demnach bietet sich das Design der Fallstudie an. Als Fallstudie bezeichnet man eine holistische Untersuchung, die ein aktuelles Phänomen in seiner natürlichen Umgebung betrachtet. Unter holistisch versteht man dabei, dass durch die Sammlung und Analyse von detaillierten Daten, die aus unterschiedlichen Quellen wie Interviews, Dokumenten und Berichten stammen, ein ganzheitliches Bild gewonnen wird. Gemäss Yin (2003) bietet sich ein Fallstudie generell an,

„when ,how' or ,why' questions are being posed, when the investigator has little control over events, and when the focus is on a contemporary phenomenon within some real-life context.“ (Yin 2003, S. 1)

Als Fall wird die Entwicklung des Linux-Kernels gewählt. Einerseits handelt es sich um eines der erfolgreichsten Open-Source-Projekte insbesondere im unternehmerischen Umfeld. Dies betrifft sowohl die Verbreitung als auch die Anzahl der beteiligten Personen und Unternehmen. Diese Breite ermöglichte es denn auch, für die Datenerhebung mehrere Firmen anzufragen und für die Datenpräsentation dennoch die gewünschte Anonymität gewähren zu können.⁹⁶

Der Linux-Kernel interessiert nicht primär als Fall an sich. Er soll als ein typischer Fall untersucht werden, um das „Phänomen“ der Firmenbeteiligung an Open-Source-Software besser zu verstehen. Stake (1994) nennt eine solche, auf das generelle Verständnis einer übergeordneten Fragestellung gerichtete Fallstudie instrumentell. Im Gegensatz dazu steht die intrinsische Fallstudie, die aus Interesse am einzigartigen Phänomen des konkreten Falls gewählt wird.

⁹⁶ Ursprünglich war es vorgesehen, den Fall einer Firma unter Berücksichtigung mehrerer Open-Source-Projekte zu betrachten. Allerdings konnte dafür kein Unternehmen gewonnen werden. Deshalb wurde der Fall umgekehrt konstruiert: Die Betrachtung eines Projekts aus der Sicht von mehreren Firmen.

Setzt sich eine Fallstudie aus mehreren (instrumentellen) Fällen zusammen, so nennt man dies eine multiple oder kollektive Fallstudie. Yin (2003) beschreibt dies als analytische Generalisierung im Gegensatz zur statistischen Generalisierung. Zur Beantwortung der (dritten) Frage, welchen Beitrag große IKT-Firmen für Open-Source-Software leisten – und wie dieser Beitrag konkret aussieht –, und um ein vertieftes Wissen über die Firmenbeteiligung zu erlangen, wurden im Sinne der Triangulation die Logfiles des Linux-Kernels quantitativ analysiert (Kapitel 5). Daneben wurde in dieser Studie aufgrund des eingangs erwähnten fehlenden empirischen Datenbestandes ein exploratives, qualitatives Vorgehen gewählt (Kapitel 6). Kruse charakterisiert die qualitative Forschung wie folgt:

„Qualitative Forschung

- will komplexe soziale Sachverhalte verstehen*
- rekonstruiert subjektive Deutungsmuster*
- hält das eigene Vorverständnis möglichst weit und lange zurück*
- versteht Deutungen und subjektive Sichtweisen*
- gestaltet sich nach dem Prinzip der Offenheit*
- offene Fragen, die Antworten sind Texte*
- kleine Stichproben“ (Kruse 2006, S. 5)*

Um die Exploration möglichst offen und kreativ zu gestalten, wird der pragmatische Ansatz, wie er von Kathleen M. Eisenhardt in „Building Theories from Case Study Research“ (1989) dargelegt wurde, gewählt. Dieser Forschungsansatz zeichnet sich durch sein iteratives Vorgehen und die starke Bindung an die empirischen Daten aus und ermöglicht somit einen kontinuierlich vergleichenden Prozess zwischen Datenerhebung und -analyse. Des Weiteren zeichnen sich die aus diesem Forschungsansatz resultierenden Theorien meist durch Neuartigkeit, Überprüfbarkeit sowie empirische Gültigkeit aus.

4.2 Gütekriterien qualitativer Forschung

Ob quantitativ oder qualitativ, jegliche seriöse Forschung muss in Bezug auf die verwendeten Verfahren wissenschaftlich anerkannte Kriterien erfüllen. Die wichtigsten Kriterien sind Signifikanz, Kompatibilität von Theorie und Beobachtung, Generalisierbarkeit, Konsistenz, Reproduzierbarkeit, Präzision und Verifikation (Strauss und Corbin 1996). Lincoln und Guba

(1985) subsumieren die von ihnen definierten Gütekriterien unter dem Begriff „Trustworthiness“ (Vertrauenswürdigkeit). Unter Berücksichtigung mehrerer Autoren können die Beurteilungskriterien von qualitativer Forschung wie folgt skizziert werden:

1. Da der Forscher explizit als beeinflussender Teilnehmer betrachtet wird, müssen die Kriterien einerseits durch die Person des Forschers (Bekanntheit, Erfahrung im Feld, Sensibilität) gewährleistet werden (Altheide und Johnson 1994). Andererseits kann durch Methoden wie Triangulation (Denzin 1989) oder die Wahl von Extrem- und Kontrastfällen (Strauss und Corbin 1996) die angestrebte Güte in der Datenerhebung erreicht werden.
2. Die Angemessenheit des Forschungsprozesses zeichnet sich aus durch die Indikation des qualitativen Vorgehens, der Methodenwahl, der Transkriptionsregeln, der Samplingstrategie sowie der Komplementarität der Einzelentscheidungen (Steinke et al. 2000). Gelegentlich wird auch der Begriff der „Gegenstandsangemessenheit“ verwendet.
3. Die empirische Verankerung der Forschungsergebnisse bedeutet, dass die Bildung sowie die Überprüfung der Theorien in den Daten begründet sein muss, wobei sowohl Verifikation als auch Falsifikation möglich ist (Steinke et al. 2000).

Zwar finden sich in den oben aufgeführten Punkten die erwähnten allgemeinen wissenschaftlichen Kriterien wieder, insbesondere zwei Begriffe führen aber immer wieder zu Methodendiskussionen und sollen deshalb in der Folge noch etwas detaillierter erläutert werden. Während bei quantitativen Methoden die Generalisierbarkeit der Forschungsergebnisse mit repräsentativen Stichproben möglich ist, bedeutet die Generalisierbarkeit bei der qualitativen Forschung, dass die Ergebnisse auf die bestimmten, erforschten Situationen bezogen verallgemeinerbar sind (Stake 1994). Durch ein systematisches und möglichst umfassendes Sampling können mehr Variationen entdeckt und die Generalisierbarkeit kann somit verbessert werden (Strauss und Corbin 1996).

Die Forderung nach Reproduzierbarkeit ist eine Forderung nach Überprüfbarkeit. Allerdings kann diese Validierung nicht nur durch Reproduktion erreicht, sondern auch durch eine transparente Nachvollziehbarkeit der Forschung gewährt werden. Steinke (2000) nennt dies die intersubjektive Nachvollziehbarkeit anstelle der intersubjektiven Überprüfbarkeit. Diese unter-

schiedliche Herangehensweise begründet sich in der Erkenntnis, dass sich z. B. kein Interview, das nicht vollständig standardisiert ist, identisch wiederholen lässt, nicht durch denselben Interviewer, und schon gar nicht durch einen anderen Forscher.⁹⁷

⁹⁷ Eine ganz andere Frage ist, ob sich bei quantitativer Forschung und standardisierten Interviews diese Replikation tatsächlich gewährleisten lässt.

5 Quantitative Erhebung der Linux-Kernel-Logfiles

„The Linux kernel is one of the most popular Open Source development projects, and yet not much attention has been placed on who is doing this development, who is sponsoring this development, and what exactly is being developed.“
(Kroah-Hartman 2007, S. 239)

5.1 Ausgangslage

Bei jedem Release des Linux-Kernels, also alle zwei bis drei Monate, werden von Jonathan Corbet mit Unterstützung von Greg Kroah-Hartman aktuelle Statistiken generiert und auf lwn.net publiziert und kommentiert.⁹⁸ Dabei werden neben einigen Totalzahlen die aktivsten Entwickler und Firmen nach „changesets“, also in sich schlüssigen Änderungen, die gemäß der Richtlinien möglichst granular sein sollten, sowie nach „changed lines“⁹⁹ aufgelistet. Zudem werden auch die „sign-offs“ ausgewertet, die als „Gatekeeper“ angesehen werden, eine etwas freiere Definition als „Maintainer“.

Die Erstellung der Statistiken ist dabei weitgehend durch das Skript gesteuert. Einerseits bietet das „Git“, die von Linus Torvalds speziell für die Linux-Community geschriebene Repository-Software, Kommandos zur Erstellung von Logfiles, die die benötigten Daten standardisiert enthalten. Andererseits werden die Statistiken darauf aufbauend von Python¹⁰⁰-Scripts generiert. Für die Zuteilung der Entwickler zu den Firmen gibt es eine Liste. Ganz im Sinne der Freien Software stehen die Programme unter der GPL und somit öffentlich zur Verfügung, wobei ein Teil der Entwickler-Firmen-Zuteilung auf Wunsch der Entwickler „privat“ gehalten wird.

98 Siehe etwa <http://lwn.net/Articles/264440/> [10.08.2009], wo auch die Top-Entwickler und -Firmen für das Jahr 2007 aufgelistet sind.

99 Die „changed lines“ enthalten sowohl die effektiv abgeänderten als auch die hinzugefügten und gelöschten Codezeilen. Lediglich umbenannte Files, die grundsätzlich als gelöschte (alter Filename) und hinzugefügte (neuer Filename) Zeilen gezählt werden, werden dabei herausgefiltert. Dies kann durch einen Parameter bei der Erstellung der Logfiles sehr einfach erreicht werden.

100 Python (<http://www.python.org/> [12.08.2009]) ist eine sogenannte Scriptsprache, die den Anspruch auf Einfachheit und Übersichtlichkeit hat und offen für verschiedene Programmierparadigmen ist. Sie ist insbesondere in der Open-Source-Community sehr beliebt.

Wie jegliche Softwaremetrik können auch diese Statistiken der Realität nur nahekommen. Einige (bekannte) Probleme dabei sind:

1. Ein „changeset“ bzw. die Anzahl der geänderten Zeilen sagt nichts aus über den tatsächlichen Zeitaufwand für den geleisteten Beitrag. Näher kann man allerdings nicht kommen, außer man befragt jeden Entwickler zu jedem seiner Beiträge einzeln, was abgesehen von der wohl fehlenden Bereitschaft der entsprechenden Personen auch an dem unverhältnismäßigen Aufwand scheitern muss. Und auch dann würde man lediglich eine „gefühlte“ und keine gemessene Statistik erhalten.
2. Die Auswertungen basieren stark auf den E-Mail-Adressen. Da diese aber nur als freier Text vorhanden sind, besteht eine Unschärfe, die sich nur durch zeitintensive Handarbeit zumindest größtenteils beheben lässt. Bei einer zweimonatigen Periodizität ist dies ein kaum durchführbares Vorgehen.
3. Jeder „changeset“ kann nur einem Autor zugeschrieben werden, auch wenn mehrere daran gearbeitet haben.
4. Da nur der Hauptentwicklungszweig betrachtet wird, werden Bugfixes (Maintenance) auf stabile Releases nicht berücksichtigt.
5. Änderungen in der Anstellung innerhalb der Auswertungsperiode können aufgrund der eindimensionalen Liste nicht berücksichtigt werden. Eine echte Datenbank für die Firmenzuteilung ist jedoch nicht erwünscht, wie in einem Artikel auf lwn.net erwähnt wird.¹⁰¹
6. In einer offenen Community sind nicht alle Personen bekannt. So erscheint in der Firmenstatistik immer eine Position „(Unknown)“, die diese unbekannten Personen enthält. In den aktuellsten Auswertungen haben die von Unbekannten gelieferten Patches eine Größenordnung von etwas über 10 %¹⁰², was doch nicht zu vernachlässigen ist. Da viele Personen einen einmaligen Beitrag leisten, lässt sich diese Problematik auch nicht durch permanente Erweiterung der Liste lösen.

101 Siehe <http://lwn.net/Articles/264440/> [10.08.2009].

102 Da es sich bei den unbekanntem Beitragenden in der Regel um eher kleine Beiträge handelt, ist der Anteil der nicht bekannten Personen gar über 20 %.

7. Das „Signed-off-by“-Tag wird sehr unterschiedlich interpretiert.¹⁰³

Um in der Frage der Firmenbeteiligungen am Linux-Kernel etwas weiter zu gehen, als es mit einer puren Auflistung möglich ist, sowie um einige der oben erwähnten Probleme zumindest teilweise zu beheben, hat sich der Verfasser dazu entschlossen, eine – auf den vorhandenen Werkzeuge basierende – eigene, einmalige Auswertung vorzunehmen.

5.2 Sampling

Analysiert wurde das gesamte Jahr 2007 (1. Januar bis 31. Dezember).¹⁰⁴ Berücksichtigt wurde in erster Linie das Sourcecode-Verzeichnis (Repository) 2.6 von Linus Torvalds (wird in der Folge als „Entwicklung“ bezeichnet und in den Beschriftungen meist weggelassen), da dieses die aktuelle „offizielle“ Kernel-Entwicklung beinhaltet. Zusätzlich wurden auch die stabilen Releases der 2.6er-Linie¹⁰⁵ analysiert, um Aussagen über die Wartung machen zu können (wird in der Folge explizit als „Wartung“ bezeichnet). Für diesen Zweck wurden sämtliche Repositories¹⁰⁶, die Änderungen im Jahr 2007 enthielten¹⁰⁷, auf den eigenen Rechner kopiert¹⁰⁸, und mithilfe der in der Versionsverwaltungs-Software bereits enthaltenen Funktionalität wurde ein Logfile erstellt¹⁰⁹.

103 Siehe dazu z. B. den Kommentar unter <http://lwn.net/Articles/266004/> [10.08.2009].

104 Aufgrund der Tatsache, dass die von der Linux Foundation publizierten Berichte zum Stand der Linux-Kernel-Entwicklung von 2008 und 2009 keine nennenswerten Unterschiede aufweisen (Kroah-Hartman et al. 2008; 2009), kann angenommen werden, dass die dieser Studie zugrunde liegenden Daten von 2007 auch zwei Jahre später noch aktuelle Ergebnisse erlauben.

105 Es wird auch noch die 2.4er-Linie gewartet, dies basiert allerdings auf der „privaten“ Initiative von Willy Tarreau.

106 Eine Liste aller privaten und offiziellen Repositories findet sich unter <http://git.kernel.org/> [10.08.2009].

107 Stichdatum war der 8. Januar 2008.

108 Das Vorgehen ist im Internet sehr genau beschrieben und funktionierte problemlos (mit Ausnahme eines Repositories, bei dem ein Lesefehler auftrat). Nach mehreren Versuchen entschloss sich der Verfasser um 21 Uhr, eine E-Mail an den Verantwortlichen in den USA zu senden. Zehn Minuten später antwortete Greg Kroah-Hartman – Angestellter von Novell/Suse – bereits, dass er den Fehler reproduzieren konnte. Das Problem ging nun weiter an Willy Tarreau in Frankreich, seinerseits Mitgründer und -inhaber der Linux-Firma Exosec. Nach über zwei Stunden Fehlersuche, kurz vor Mitternacht, gab dieser auf und fragte Linus Torvalds, Angestellter der Non-Profit-Organisation Linux-Foundation in den USA, um Hilfe. Um ein Uhr morgens erhielt der Verfasser die Nachricht von Willy Tarreau, dass Torvalds den Fehler gefunden hätte und er ihn sofort korrigieren würde. Den Dank per E-Mail am folgenden Morgen um acht Uhr beantwortete er erneut innerhalb von Minuten und stellte das Geschehene als „normal“ und selbstverständlich dar. Er dankte dem Verfasser für das Auffinden und Melden des Fehlers. Diese Anekdote zeigt die offen praktizierte Arbeitsteilung sowie das große Berufsethos der Linux-Kernel-Community.

109 Das Kommando dazu lautet `git log --since=2007-01-01 --until=2007-12-31 --no-merges --no-renames -M -C --find-copies-harder --ignore-all-space --pretty=medium --numstat >log-2.6.txt`, wobei man dazu im Rootverzeichnis des jeweiligen Repositories stehen muss. Eine Beschreibung der Parameter findet sich unter

5.3 Datenerhebung

Die Struktur der Log-Daten lässt es zu, dass die Autoren und Reviewer der einzelnen Beiträge klar erkennbar sind und somit weitgehend maschinell ausgewertet werden können. Typischerweise sieht die Beschreibung eines „changesets“ wie folgt aus:

```

1  commit ed2c12f323e8fafbc94f9bcfb924f9df36e64dc7
2  Author: Andrew Morton <akpm@linux-foundation.org>
3  Date: Thu Jul 19 01:50:35 2007 -0700
4
5  kernel/sysctl.c: finish off the warning comments
6
7  I've been chasing these comments around this file all week.
8  Hopefully we're straight now.
9
10 Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
11 Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>
12
13 4      1      kernel/sysctl.c
```

Zeile 1 enthält die eindeutige Identifikation für den „changeset“. Dieser wurde benötigt, um einen Patch, der in mehreren Repositories vorhanden ist, zu eliminieren.¹¹⁰ In der zweiten Zeile steht der Autor mit Namen und E-Mail-Adresse; es wurde nur die E-Mail-Adresse extrahiert. Das Datum aus Zeile 3 sowie der folgende Kommentar (Zeilen 4-9, inkl. Leerzeilen) waren für die vorliegende Analyse nicht relevant. In den Zeilen 10 und 11 sind die Reviewer enthalten; die Zeile beginnt jeweils mit „Signed-off-by:“ bzw. „Aked-by:“ und enthält in der Folge wiederum Namen und E-Mail-Adresse. Da sich die vorliegende Analyse ausschließlich mit den Autoren beschäftigt, sind diese Informationen auch nicht von zentraler Bedeutung. Zu Kontrollzwecken wurden sie jedoch ebenfalls extrahiert und situativ ausgewertet, um allfällige Unterschiede festzustellen. In der Zeile 13 ist das geänderte Sourcefile erwähnt – in anderen „changesets“ sind durchaus auch mehrere Files aufgelistet, jedes in einer neuen Zeile. Die erste Zahl nennt die hinzugefügten, die zweite die gelöschten Zeilen, wobei eine Änderung als eine hinzugefügte und eine gelöschte Zeile erscheint. Berücksichtigt wurden hier nur die hinzugefügten Zeilen. Dies lässt sich dadurch begründen, dass nur so bei einer weitgehend maschinellen Auswertung gewährleistet werden kann, dass eine geänderte Zeile nicht doppelt in der Statistik erscheint. Zudem lässt sich argumentieren, dass das Löschen einer Zeile in den

<http://www.kernel.org/pub/software/scm/git/docs/git-log.html> [10.08.2009].

¹¹⁰ Da die Stable-Releases aus dem Repository von Linus Torvalds gezogen werden, sind sämtliche „changesets“ daraus früher oder später zumindest doppelt vorhanden.

meisten Fällen keinen wirklichen Beitrag darstellt.

5.4 Datenanalyse

Für das Prüfen sowie die Konsolidierung der Daten wurde ein eigenes Perl¹¹¹-Script geschrieben, sodass die Auswertung jederzeit wiederholt werden kann. Dies war wichtig, da häufig erst die generierten Konsolidierungen Fehler und Mängel, die es noch zu beheben galt, aufdeckten.¹¹²

In einem ersten Schritt war es notwendig, die E-Mail-Adressen manuell zu verifizieren und zu klassifizieren. Als Vorgabe diente die von Greg Kroah-Hartman geführte Liste, die für die regelmäßigen Auswertungen auf lwn.net eingesetzt wird. Bei allen darin nicht aufgeführten Adressen wurden zuerst offensichtliche Tippfehler, leicht abweichende Schreibweisen sowie Redundanzen korrigiert. Danach wurden alle Adressen dahin gehend geprüft, ob die Domain einer Firma zuzuordnen war (und ob es sich dabei nicht um einen Mailprovider handelt).¹¹³ Die verbliebenen rund 650 unbekannt Adressen wurden in einer individuell adressierten E-Mail angeschrieben, worauf rund 200 antworteten und somit klassifiziert werden konnten.¹¹⁴ Dies führte dazu, dass die hier vorliegende Analyse mit gut 18 % unbekannt Autoren und 7,8 % Patches respektive 5,8 % beigefügten Codezeilen von unbekannt Autoren deutlich unter den entsprechenden „offiziellen“ Statistiken von lwn.net liegt.

Für die Kategorisierung der Firmen wurde wie in der MERIT-Studie (Ghosh 2006) die Amadeus-Datenbank¹¹⁵ verwendet (Stand Juli 2008). Da sich der Verfasser auf den freien Teil der Datenbank beschränken musste, standen lediglich die Informationen zu Ort, Sektor und Größe

111 Perl (<http://www.perl.org/> [12.08.2009]) ist wie Python eine Scriptsprache, die mehrere Programmierparadigmen unterstützt. Die Sprache ist sehr mächtig, jedoch aufgrund ihrer großen Flexibilität und kryptischen Syntax schwer erlernbar und für Außenstehende auch oft kaum verständlich.

112 Die so ermittelten Detaildaten wurden recht umfangreich und sind in je einer OpenOffice-Calc-Datei für die Entwicklung und Wartung detailliert aufgelistet. Sie können bei Bedarf beim Autor angefordert werden.

113 Nicht zuletzt aus arbeitsrechtlichen Gründen ist es meist eine Vorgabe der Firmen, dass Beiträge in Open-Source-Projekten zwingend mit der Firmen-E-Mail-Adresse gepostet werden. Eine Firma geht sogar so weit, dass auch Beiträge, die in der Freizeit geleistet werden, mit der Firmen-E-Mail zu kennzeichnen sind. Dass sich nicht alle Entwickler daran halten, spricht eher für deren Reputation als gegen die Gültigkeit der Regel.

114 Die E-Mail-Anfrage findet sich im Anhang E.

115 Zu finden unter <http://www.bvdep.com/en/Company%20data%20-%20international.html> [10.08.2009]. Es stand dabei lediglich das frei zugängliche Verzeichnis zur Verfügung, welches nur sehr rudimentäre Informationen enthält.

der Firmen zur Verfügung. Die Sektoren sind dabei in Bezug auf Informatik nur sehr rudimentär. Konkret konnte nur nach Hardware, Softwareentwicklung und Dienstleistungen, Telekom (inkl. ISPs) sowie Handel unterschieden werden. Es wäre durchaus wünschenswert gewesen, die Kategorisierung detaillierter durchzuführen. Aufgrund fehlender Daten – vor allem bei kleineren Firmen – wurde jedoch darauf verzichtet. Leider ist nirgends genau erklärt, wie die Firmengröße definiert ist.

Die Firmen mussten manuell in der Datenbank gesucht werden. Da es in der Regel mehrere Treffer gab, musste der gewünschte ausgewählt werden, was eine mögliche Fehlerquelle war. Um die Fehlerquote möglichst gering zu halten, wurde bei Zweifeln jeweils die Homepage der Firma konsultiert. Es wurde immer die Muttergesellschaft erfasst, auch wenn erkennbar war, dass der Autor in einer Zweigstelle beschäftigt war.

Ungefähr 10 % der Firmen konnten in der Datenbank nicht gefunden werden. Bei diesen wurde versucht, die entsprechenden Informationen im Internet (Homepage, Domainbesitzer, Suche mittels Google) zu erhalten. So konnte in jedem Fall das Land ermittelt werden. Standen keine Informationen zum Sektor und zur Firmengröße zur Verfügung, wurde angenommen, dass es sich um sehr kleine Firmen im Bereich der Softwareentwicklung und Dienstleistungen handelt.

6 Qualitative Experteninterviews von Linux-Kernel-Entwicklern in großen IKT-Firmen

6.1 Das Experteninterview

Interviews als Datenerhebungsform sind in der qualitativen Forschung am weitesten verbreitet und haben eine fast hundertjährige Geschichte (Fontana und Frey 1994; Platt 2001). Je nach Forschungsfrage werden spezifische Formen gewählt. Kruse (2006) listet neben den „klassischen“ Formen des narrativen, fokussierten, problemzentrierten, ethnografischen sowie des Leitfaden- und Experteninterviews und der Gruppendiskussion zehn weitere Varianten auf.

Obwohl der Begriff „Experte“ – wie weiter unten aufgeführt – einer Differenzierung bedarf, wird dennoch die Methode des Experteninterviews dem problemzentrierten (Witzel 2000) oder dem fokussierten Interview (Merton et al. 1990; Hopf 2000) vorgezogen, denn Experteninterviews erlauben im Vergleich mehr Offenheit und entsprechen daher dem explorativen Charakter der Studie besser.

Bogner und Menz (2002a) unterscheiden zwischen explorativem, systematisierendem und theoriegenerierendem Experteninterview. Die vorliegende Arbeit orientiert sich an letzterem, *„an dessen Endpunkt idealerweise die Formulierung einer ‚formalen‘ Theorie steht.“* (Bogner und Menz 2002a, S. 38 f.). Diese „formale“ Theorie ist idealerweise die *„Generalisierung einer Typologie“* (Bogner und Menz 2002a, S. 38) und basiert auf der subjektiven Sichtweise der Experten, sie kann und soll folglich nicht als objektive Realität verstanden werden. Die Vergleichbarkeit der erhobenen Daten soll trotz der intendierten Subjektivität der Aussagen mit einem Leitfaden unterstützt werden. Zudem muss empirisch eine gemeinsame organisatorische bzw. institutionelle Anbindung der befragten Experten erreicht sein (Bogner und Menz 2002a).

6.1.1 Wer ist ein Experte?

Wer als Experte im Sinne des Experteninterviews gilt, wird widersprüchlich definiert (Bogner

und Menz 2002a; 2002b; Meuser und Nagel 2002b). Einigkeit besteht darin, dass Experten im Sinne des Experteninterviews keinen gesellschaftlichen Elitestatus innehaben oder externe Gutachter sind, sondern dass sich das Expertentum aus dem Forschungsinteresse ableitet, d. h., dass die Forschenden der interviewten Person den Expertenstatus zuweisen.¹¹⁶ Zu Beginn einer Studie kann nicht abschließend festgelegt werden, wer im gegebenen Fall als zu befragender Experte gilt (Bogner und Menz 2002b). So hat sich bei der Befragung der Softwareentwickler gezeigt, dass auch Personen außerhalb der Entwicklungsabteilung (Koordinatoren, Projektleiter, Juristen) interviewt werden sollten.

6.1.2 Rolle des Interviewenden

Zur Rolle der Interviewenden im Gespräch unterscheiden Bogner und Menz (2002b) sechs (Ideal-)Typen: (1) Co-Experte, (2) Experte einer anderen Wissenskultur, (3) Laie, (4) Komplize, (5) potenzieller Kritiker und (6) Autorität. Während die beiden letzten Rollen problematisch erscheinen (in der Regel wird der Befragte eine defensive Haltung einnehmen) und nicht angewendet werden sollen, können die anderen vier Rollen durchaus variabel in den Interviews eingenommen werden, es kann sogar mit ihnen „gespielt“ werden. In der vorliegenden Studie wurde von einer möglichst neutralen¹¹⁷ Laienposition des Interviewenden ausgegangen, um dann situativ als Co-Experte, Experte einer anderen Wissenskultur oder gar als Komplize aufzutreten.

Für die Einflussnahme des Interviewenden sei erwähnt, dass es kaum möglich ist, die eigene Person im Interview als vollständig neutral zu betrachten. Einerseits beeinflusst sowohl die Person wie die Art der Kommunikation den Interviewten, andererseits ist ein Interview immer eine Kommunikation zwischen Subjekten und deshalb subjektiv (Foddy 2003). Deshalb muss

116 „Methodologisch gesehen bestimmt sich der ExpertInnenstatus einer Person in Relation zum jeweiligen Forschungsinteresse; eine Person wird zur ExpertIn gemacht, weil wir wie auch immer begründet annehmen, dass diese Person über ein Wissen verfügt, das ihr zwar nicht unbedingt alleine verfügbar ist, das aber doch nicht jedermann bzw. jederfrau im interessierenden Handlungsfeld zugänglich ist. Als ExpertIn wird mithin angesprochen,

- wer in irgendeiner Weise Verantwortung trägt für den Entwurf, die Implementierung oder die Kontrolle einer Problemlösung, oder

- wer über einen privilegierten Zugang zu Informationen über relevante Personengruppen, Sozallagen und Entscheidungsprozesse verfügt.“ (Meuser und Nagel 2002a, S. 259)

117 Eine „durchgezogene“ Neutralität wäre in der gegebenen Situation wohl eher unglaubwürdig, da insbesondere aufgrund des akademischen Status des Forschers ein fundiertes Wissen in der Materie durch die Befragten vorausgesetzt werden durfte.

der Einfluss des Interviewers reflektiert und bei der Analyse berücksichtigt werden:

„Natürlich genügt es, ein einziges Interview geführt zu haben, um zu wissen, wie schwierig es ist, seine Aufmerksamkeit pausenlos auf das, was gerade gesagt wird (und nicht nur in Worten), zu konzentrieren und sich ständig im voraus Fragen zu überlegen, die sich ‚natürlich‘ in das Gespräch einfügen, zugleich aber einer Art theoretischer ‚Linie‘ zu folgen. Das heißt, niemand kann sich vor dem Effekt des Aufdrängens einer Problematik in Sicherheit wiegen, den naiv-egozentrische oder einfach nur unkonzentrierte Fragen hervorbringen können, und vor allem auch nicht vor der Rückwirkung, die auf diese Weise dem Befragten ‚in den Mund gelegte‘ Antworten auf den Forscher haben können, der in seiner Interpretation unter Umständen ein Artefakt, das er selbst, ohne es zu merken, produziert hat, für bare Münze nimmt.“ (Bourdieu 1997, S. 782)

Es gilt demnach, die Interviewtechnik zu verfeinern. Dies geschieht unter anderem im iterativen Vorgehen der Datenerhebung und -analyse. Das bedeutet auch, dass der Leitfaden soweit nötig angepasst werden kann. Dies wiederum erfordert eine Anpassung der Interviewführung.

6.2 Sampling

Das Ziel von qualitativer Forschung ist die „Rekonstruktion typischer Muster“ (Helfferich 2005, S. 153). Um eine Verallgemeinerbarkeit der Ergebnisse zu erreichen – denn nur so kann von empirischen Daten, im Gegensatz zur Illustration, gesprochen werden –, werden durch Bestimmung der interessierenden Attribute (Stake 1994) die Eckpunkte festgelegt, die den zu untersuchenden Fall repräsentieren (siehe Abb. 11).

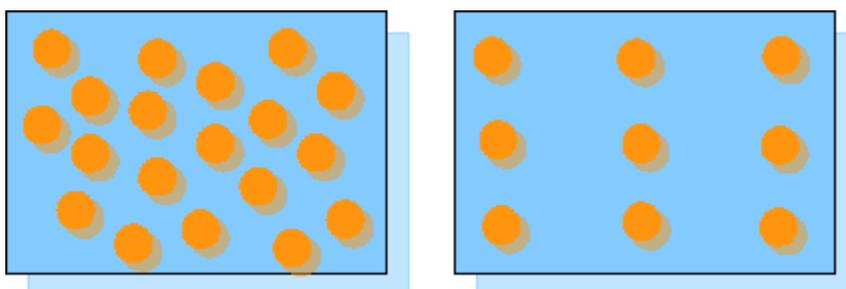


Abbildung 11: Vergleich repräsentatives und theoretisches Sampling (nach Kruse 2006, S. 37 f.)

Bei einem explorativen Vorgehen wird diese Praxis „theoretical sampling“ (Glaser und Strauss 1967) genannt. Eisenhardt (1989) schließt dabei methodisch eine Zufallsstichprobe nicht grundsätzlich aus, legt aber Wert darauf, dass aufgrund der meist relativ kleinen Stichprobe möglichst kontrastierende Fälle zu wählen sind, um die in den Daten begründeten Kategorien konstruieren zu können.

In dieser Studie erfolgte eine „sekundäre[.] Selektion“ (Merkens 2000, S. 288), d. h. eine „Gelegenheitsstichprobe mit beschränktem (theoriegeleiteten) Einfluss“. Einerseits war die Anzahl der potenziell zu Befragenden aus geografischen Gründen limitiert. Andererseits war die Zugänglichkeit der Personen innerhalb der Firmen eingeschränkt. Um diese Problematik zu mindern, wurden Befragungen in mehreren Firmen durchgeführt („multiple-case studies“, Yin 2003).

6.3 Rekrutierungsstrategie

Für die Rekrutierung standen zwei Strategien zur Auswahl: über die Community oder über die Firmen. Aufgrund der Tatsache, dass im Interview zumindest ansatzweise Betriebsinterna bzw. -geheimnisse zur Sprache kommen könnten, bestand bei einer Anfrage über die Community die Gefahr, dass als „heikel“ eingestufte Themen ausgeblendet würden. Andererseits war durch eine Rekrutierung über die Firmen die Auswahlmöglichkeit kleiner, da nur Mitarbeitende von Firmen zur Disposition stünden, die mit dem Forschungsprojekt einverstanden waren. Zudem würde die Selektion der tatsächlichen Interviewpartner durch Firmenvorlieben eingeschränkt.

Die Entscheidung fiel für die Rekrutierung über die Firmen, weil die Bedenken zur Zurückhaltung bei den Antworten als gewichtiger beurteilt wurden als die eingeschränkte Auswahl von Interviewteilnehmenden. Zudem wurde die Rekrutierung über die Community als Option offengelassen, falls eine ungenügende Anzahl und Differenzierung des Samples ersichtlich würde. Diese Option erwies sich allerdings nicht als notwendig. Die beteiligten Firmen haben eine Vertraulichkeitserklärung vom Verfasser erhalten.¹¹⁸

118 Die Vorlage der Vertraulichkeitserklärung auf Deutsch und Englisch findet sich im Anhang B. Punktuelle Anpassungen mussten auf Wunsch bei einer der Firmen noch vorgenommen werden. Diese Änderungen sind weniger inhaltlicher als formaler Art – ein deutliches Zeichen dafür, dass das Dokument durch die

Um eine möglichst große Homogenität bei den einbezogenen Firmen zu erreichen, wurden nur Firmen für eine Beteiligung angefragt, die dem IT-Sektor zuzurechnen sind, die mindestens zehn Linux-Entwickler beschäftigen, insgesamt mehr als Tausend Beschäftigte haben, in Europa präsent sind und nicht zur New Economy gezählt werden, also eine mindestens zwanzigjährige Firmengeschichte aufweisen.¹¹⁹ Zudem wurden primär Interviewpartner aus Deutschland gesucht. Dadurch konnten kulturelle und juristische Differenzen vermieden werden, um die Analyse bei einem doch relativ kleinen Sampling zu erleichtern.¹²⁰

Die Rekrutierung der Firmen verlief unterschiedlich. Alle sechs infrage kommenden Unternehmen wurden mit einem offiziellen Schreiben des Betreuers der Dissertation, Prof. Dr. Bernd Lutterbeck, für die Teilnahme von Mitarbeitenden angefragt (siehe Anhang A). Nach zum Teil mehrmaligem Nachfragen sowie auch begleitendem „Nachhaken“ über das berufliche Netzwerk des Verfassers konnte mit einer Ausnahme ein Kontakt aufgebaut werden, woraufhin sich letztlich fünf Firmen für eine Teilnahme bereit erklärten. Wichtig bei diesem Prozess war, in den Firmen einen Interessierten zu gewinnen, da nur auf diesem persönlichen Weg die Rekrutierungsstrategie Erfolg versprach.

Durch die betriebliche Kontaktperson („gatekeeper“¹²¹) wurden nach Abwicklung der administrativen Arbeiten (vornehmlich rechtliche und organisatorische Fragen) potenzielle Interviewpartner durch die Kontaktperson in der Firma schriftlich oder mündlich angesprochen. Die Mitarbeitenden konnten sich danach direkt beim Verfasser melden. Drei der fünf Firmen wollten jedoch die Interviewpartner selbst bestimmen, was respektiert wurde. Bei den anderen beiden Firmen wurde in der Folge versucht, nach der Schneeballmethode über die zuerst interviewten gezielt „Fälle“ anzusprechen, was allerdings als Nachteil zu einer zumindest teilweise „geklumpten Stichprobe“ (Merkens 2000, S. 293) führte. Da die gewährte Anzahl der Interviews pro Firma bei maximal fünf lag, waren jedoch die Möglichkeiten stark eingeschränkt.¹²²

Rechtsabteilung begutachtet wurde.

119 Somit wurden Firmen wie Red Hat oder Google bewusst ausgeschlossen.

120 Da es möglich war, je ein Interview in den USA und der Schweiz durchzuführen, wurden diese im Sinne von Kontrastfällen ebenfalls berücksichtigt.

121 Es hat sich gezeigt, dass die Rekrutierung nur dann erfolgreich war, wenn der Gatekeeper selbst ein großes Interesse am Thema Open Source hatte. Die hierarchische Stellung innerhalb der Firma war für den Zugang nicht entscheidend. Es hatte sich jedoch gezeigt, dass bei einer höheren Position der Prozess schneller vorstatten ging.

122 Es hat sich herausgestellt, dass die Anzahl der gewährten Interviews bei denjenigen Firmen, die den Gesprächspartner selbst wählen wollten, mit einem bis drei Interviews deutlich niedriger war. Zudem konnten

Von einer theoretischen Sättigung kann insofern gesprochen werden, als sich in Bezug auf die formale Organisation und die formalen Prozesse der Linux-Entwicklung die von den Interviewten erhaltenen Informationen weitgehend deckten. Hingegen wären hinsichtlich der individuellen Arbeitssituation eine größere Anzahl an Interviews durchaus wünschenswert gewesen. Da die Erreichung dieses Zieles allerdings höchstens um den Preis einer größeren Heterogenität¹²³ der Firmen im Sample erreichbar gewesen wäre, wurde darauf verzichtet.

6.4 Leitfadenentwicklung

Im Sinne einer explorativen Studie wurden die Interviews durch einen Leitfaden unterstützt, jedoch ließ dieser genügend Raum für Offenheit. Demnach wurde eine Mischform zwischen narrativem und strukturiertem Interview gewählt bzw. ein teil- oder halbstandardisiertes (Flick 2002) respektive teilnarratives (Helfferrich 2005) Interview durchgeführt. Ein Leitfaden für diese Interviewform enthält in der Regel mehrere Themenblöcke, die jeweils durch eine offene Frage eingeleitet werden. Zu jedem Thema können in der Folge konkrete, theoriegeleitete und durchaus auch konfrontative oder reflexive Nachfragen gestellt werden.

Der Leitfaden wurde anhand des SPSS-Prinzips erstellt (Helfferrich 2005):

1. **Sammeln** von möglichst vielen Fragen (Brainstorming)
2. **Prüfen** der Fragen hinsichtlich der Offenheit, Subjektivität, Neuheit und Relevanz mit einer damit einhergehenden Reduktion eines vermutlich großen Teils der Fragen
3. **Sortieren** und Bündeln in nicht zu viele Themenblöcke
4. **Subsumieren** der einzelnen Fragen eines Themenblocks unter einer möglichst offenen, erzählgenerierenden Einstiegsfrage

Die Interviewpartner haben Übung darin, sachlich und präzise zu kommunizieren. Da die Interviewsituation eine Distanzierung von der Alltagsrealität ermöglicht, beginnt der Leitfa-

bei beiden Firmen, die eine freie Wahl zuließen, jeweils sogar sechs anstatt der gewährten fünf Interviews durchgeführt werden.

123 Wie bereits an einer früheren Stelle erwähnt, bezieht sich diese unerwünschte Heterogenität vor allem auf den Standort und die damit möglicherweise verbundenen Unterschiede im Arbeitsrecht und in der Kultur, aber auch auf andere Branchen.

den¹²⁴ mit einer relativ geschlossenen Frage nach der Ausbildung und wird danach zunehmend offener (siehe dazu z. B. Trinczek 2002). Es folgen Fragen zum beruflichen Alltag, den Differenzen von Open Source zu Closed Source sowie der speziellen Problematik der Entwicklung von Open-Source-Software im kommerziellen Umfeld. Abschließend wird nach der Einschätzung der aktuellen beruflichen Situation und nach der realistischen sowie idealen beruflichen Zukunft gefragt.

Der Leitfaden wurde während der Datenerhebungsphase punktuell angepasst, um neue Erkenntnisse einfließen zu lassen und das Erkenntnisinteresse präzisieren zu können. Es wurde jedoch darauf geachtet, dass das Gerüst des Leitfadens nicht verändert wurde, um die Analyse nicht zu strapazieren.

Als Vorbereitung zur Ausarbeitung des Leitfadens für Entwickler wurde in drei Firmen je ein offenes Interview mit einem „Kordinator“ geführt, um einerseits das Projekt vorzustellen wie zu besprechen und andererseits um erste Informationen zur Organisation und Struktur des Betriebes zu gewinnen (Froschauer und Lueger 2002).

Zusätzlich wurden zwei Interviews mit dem Verfasser bekannten Softwareentwicklern aus dem proprietären Umfeld durchgeführt, allerdings mit einem modifizierten Leitfaden. Dabei war nicht der Test des Leitfadens das Ziel, sondern das Herantasten an die Interviewsituation sowie das Testen des Aufnahmegeräts. Auf einen Pretest wurde verzichtet, da sich bei einem iterativen Vorgehen ohnehin Änderungen des Leitfadens ergeben können.

6.5 Datenerhebung

Fallstudien enthalten meist mehrere Methoden der Datenerhebung wie Interviews, Fragebogen oder Beobachtungen sowie Recherchen in Archiven (Eisenhardt 1989). Für die Erhebung der qualitativen Daten wurde in der vorliegenden Studie vorwiegend mittels Interviews gearbeitet. In den öffentlichen Archiven des Linux-Kernels sowie in den Newsportalen wurden jedoch die im Sampling enthaltenen Firmen aktiv verfolgt. Des Weiteren wurden zahlreiche Fir-

124 Der gesamte Leitfaden auf Deutsch und Englisch findet sich im Anhang C. Abgesehen von zwei Ausnahmen konnte der deutsche Leitfaden verwendet werden. In den restlichen beiden Interviews wurde der englische Leitfaden verwendet.

menberichte und -präsentationen mit Blick auf die definierte Forschungsfrage durchgesehen. Diese sekundären Informationen bildeten einen wichtigen Hintergrund für die Analyse und verhalfen im Sinne der Triangulation zu einer erhöhten Validität. Kern der Daten bleiben jedoch die Interviews.

Die Datenerhebung wurde in der Zeit von September 2007 bis Juni 2008 durchgeführt. Neunzehn Interviews konnte der Verfasser persönlich vor Ort bei den Interviewteilnehmenden führen; fünf Gespräche wurden telefonisch durchgeführt. Aufgrund der zum Teil längeren Anfahrtswege wurden dabei meist mehrere Interviews pro Tag eingeplant.

Die Dauer der Interviews war sehr unterschiedlich:

	Minuten
Gesamtdauer	934
Längstes Interview	104
Kürzestes Interview	28
Durchschnittliche Interviewdauer	54,94

Tabelle 4: Dauer der Interviews

Die Interviews wurden dabei mit dem „Handy Recorder H4“ digital als MP3 aufgenommen. Nach dem Gespräch wurde direkt mit den Interviewteilnehmenden ein Datenblatt (siehe Anhang D) mit soziodemografischen und betrieblichen Angaben ausgefüllt. Zudem wurden die wichtigsten (subjektiven) Erkenntnisse in einem Interviewprotokoll zeitnah festgehalten (Helfferich 2005).

Es hat sich gezeigt, dass die Gesprächspartner in Bezug auf Interna sehr offen waren, wohl nicht zuletzt durch die „offizielle“ Sanktionierung des Gesprächs durch das Management. Andererseits kam bei einigen wenigen Gesprächen der Eindruck auf, dass sich die Interviewten vorsichtig ausdrückten, wenn es um ihre Position zur Firma ging.¹²⁵

6.6 Datenanalyse

Unabhängig von der jeweiligen Methode können die Analysearten oder -schritte als „selektie-

¹²⁵ Typisch war dabei der Kontrollblick zur Tür, ob diese auch tatsächlich geschlossen war.

ren“, „codieren“, „gruppieren“ und „interpretieren“ zusammengefasst werden (Bernard 2000). Dabei werden die Schritte nicht streng sequenziell abgearbeitet, sondern kontinuierlich und repetitiv, um so zusätzliche Tiefe zu ermöglichen (siehe Abb. 12).

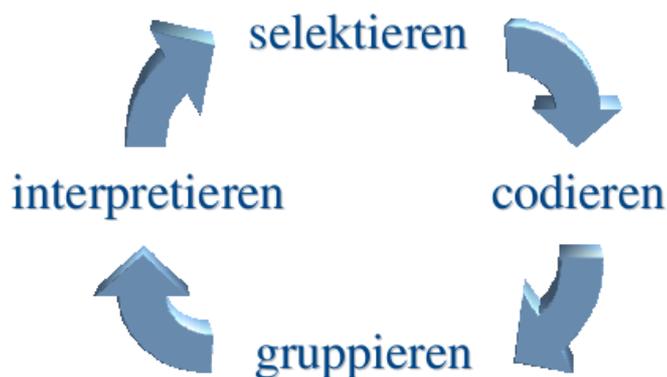


Abbildung 12: Analyseprozess (nach Bernard 2000)

Mit diesen Analyseschritten wird nicht erst nach der beendeten Datenerhebung begonnen, sondern gewonnene Erkenntnisse, Konzepte, Theorien und Typologien fließen in die weiteren Interviews ein und erlauben so eine Zuspitzung und Vertiefung des Forschungsgegenstandes.

6.6.1 Transkription

Alle siebzehn Interviews mit den Linux-Kernel-Entwicklern wurden vollständig transkribiert. Die restlichen sieben Interviews mit den Evangelisten¹²⁶, Projekt- und Produktmanagern wurden zwar mit Ausnahme von zwei auch digital aufgenommen, jedoch lediglich zusammengefasst, da sie als Hintergrundinformationen dienten und deshalb nicht vollständig bzw. sequenziell codiert wurden. Verwendet wurde die freie Software „Transcriber“¹²⁷, die beinahe optimal den Bedürfnissen des Verfassers entsprach.¹²⁸

Sämtliche Transkripte wurden durch den Forscher selbst erstellt, was sich als sehr sinnvoll herausstellte. Auch wenn das Transkribieren mitunter eine monotone und sehr zeitintensive Arbeit ist, so fand doch während der Verschriftlichung ein kreativer Prozess statt, der für die

126 Der Begriff „(Technologie-)Evangelist“ ist eine in der Informatik häufig verwendete Bezeichnung für eine Person, deren Job es ist, (neue) Technologien zu fördern (siehe dazu <http://www.gnote.ws/> [20.08.2009]).

127 Eine detaillierte Beschreibung des Programms findet sich im Anhang G.

128 Siehe dazu die Beschreibung des Programms in Anhang G.

Analyse sehr wertvoll war. Zudem konnten durch das Transkribieren ein Überblick sowie eine Vertrautheit mit den Texten gewonnen werden, die beim Codieren erneut sehr hilfreich waren.

Für die Transkription wurden vereinfachte Regeln angewendet (Flick 2002). Dies bedeutet, dass zwar jeweils ein vollständiges Transkript erstellt wurde, dass aber auf die Verschriftlichung der linguistischen Eigenheiten sowie der nonverbalen Kommunikation weitgehend verzichtet wurde, da dies für das weitere Vorgehen nicht von Interesse war. Daraus resultierte ein inhaltlich vollständiger, sich weitgehend an der Standardorthografie (Kowal und O'Connell 2000) orientierender und gut lesbarer Text.

Die Transkripte wurden nicht anonymisiert, da sie ausschließlich vom Verfasser bearbeitet wurden. Dies ist insofern vorteilhaft, als Namen, Projekte und Orte in den Rohdaten der Realität entsprechen und so in der Analyse das Bild der realen Umgebung der Interviewteilnehmenden vermitteln. Allerdings wurden zur Gewährung der versprochenen Anonymität die in der vorliegenden Dissertationsschrift zitierten Textstellen anonymisiert. Als Konsequenz dieses Vorgehens können die Transkripte von Dritten nicht eingesehen werden.

6.6.2 Induktives Codieren

Die Transkripte werden mittels Inhaltsanalyse in einem systematischen, regelgeleiteten Vorgehen (Mayring 1997) mit der freien Software WeftQDA¹²⁹ codiert. Da es sich hier um eine explorative Studie handelt, wurde eine induktive Kategoriendefinition gewählt. Das heißt, dass die Kategorien nicht aus der Theorie, sondern aus dem vorliegenden Datenmaterial abgeleitet wurden. Dieses Vorgehen lehnt sich stark an das offene Codieren der Grounded Theory von Glaser und Strauss an (Mayring 1997).

Aufgrund der Fragestellung findet bereits eine Vorselektion statt, welches Textmaterial für eine Kategorisierung hinzugezogen werden soll. Im Prozess des Codierens wird jedes Transkript Zeile für Zeile durchgearbeitet. Es können jedoch jene Textpassagen uncodiert belassen werden, die nur „*Unwesentliches, Ausschmückendes, vom Thema Abweichendes*“ (Mayring 1997, S. 76) enthalten.

129 Eine detaillierte Beschreibung des Programms findet sich im Anhang G.

Konkret wurde in einem ersten Durchgang das Material so lange durchgearbeitet, bis sich keine neue Kategorien mehr ergaben. Diese wurden danach erstmals genauer daraufhin überprüft, ob sie für die vorliegende Fragestellung zweckmäßig waren und ob sie noch zusammengefasst oder weiter aufgeteilt werden müssten. Falls sich hier Veränderungen ergaben, musste die Codierung wieder neu beginnen. Mit diesen Analyseschritten nähert man sich laufend einem Kategoriensystem an, in dem ausgewählte Textpassagen ein verdichtetes Thema illustrieren (Mayring 1997). Aufgrund dieses induktiv erarbeiteten Kategoriensystems können nun Typen gebildet werden.

6.7 Typenbildung

Um komplexe Realitäten erfassen, verstehen und erklären zu können, wird in vielen qualitativen Studien durch Bildung von Typen die Komplexität reduziert (Bohnsack 1992). Im Sinne von Max Weber kann so die „*chaotische Vielfalt individueller Erscheinungen*“ (nach Kaesler 2003, S. 232) geordnet werden. Dies ist vor allem dann sinnvoll, wenn man aufgrund von nicht wiederholbaren und somit nicht falsifizierbaren empirischen Daten „*das nicht Generalisierbare zu generalisieren*“ (Janoska-Bendl 1965, S. 84) versucht. Ein auf solche Weise entstandener und sehr bekannter Typus ist der „homo oeconomicus“.

Die Typen werden oft als Ideal- (Weber 1951; Gerhardt 2001) oder Prototypen (Mayring 1997) bezeichnet; ein Typus ist ein theoretisches Konstrukt, „*d. h. seine Elemente stammen aus der Wirklichkeit, er beschreibt aber nicht die Wirklichkeit*“ (Janoska-Bendl 1965, S. 25), er ist demnach in der Realität nicht anzutreffen. Dies ergibt sich daraus, dass die Typen auf Merkmale reduziert werden, die innerhalb des Typs möglichst homogen (interne Homogenität) und in Abgrenzung zu anderen Typen möglichst heterogen (externe Heterogenität) sein sollen (Kelle und Kluge 1999), d. h., die Konstruktion ist theoriegeleitet. Die so gebildeten Typen sind jedoch nicht als Theorie zu verstehen, sondern sollen in erster Linie zur Hypothesengenerierung und somit als Grundlage oder als Zwischenschritte zur Theoriebildung dienen (Kelle und Kluge 1999), und zwar aufgrund von empirisch gewonnenen und empirisch überprüfbareren Daten (Janoska-Bendl 1965).

Nach Kelle und Kluge (1999) lässt sich der Prozess der Typenbildung, der als ein heuristisches Instrument (Janoska-Bendl 1965) zu verstehen ist, in vier Teilschritte unterteilen, wobei

es sich hier nicht um ein starres oder lineares Schema handelt (siehe Abb. 13).

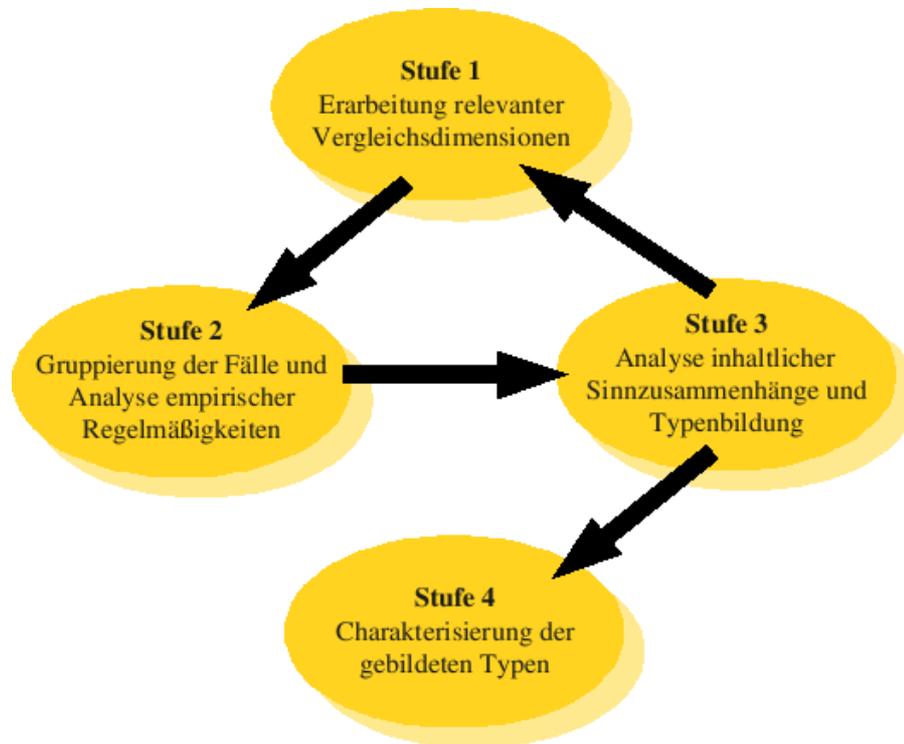


Abbildung 13: Prozess der Typenbildung (nach Kelle und Kluge 1999, S. 82)

In einem ersten Schritt gilt es, die Dimensionen zu finden und zu spezifizieren, die die interne Homogenität und die externe Heterogenität definieren. Bei einem explorativen Vorgehen geschieht dies in erster Linie durch die empirischen Daten. Im Sinne der Grounded Theory (Glaser und Strauss 1967) können diese Dimensionen anhand der Definition von Codes und Kategorien sukzessive gebildet werden und durchaus auch Einfluss auf die Fallauswahl („theoretisches Sampling“) nehmen. In der vorliegenden Studie wurde die Typenbildung jedoch erst mit der Datenanalyse begonnen.

Im Folgenden werden die Fälle „*anhand der definierten Vergleichsdimensionen und ihrer Ausprägungen gruppiert und die ermittelten Gruppen hinsichtlich empirischer Regelmäßigkeiten untersucht*“ (Kelle und Kluge 1999, S. 86). Dies geschieht, indem man die eruierten Merkmale in einer Matrix darstellt und die einzelnen Fälle den Feldern zuweist.

In einem dritten Schritt werden die Zusammenhänge der Kategorien und Fälle inhaltlich verglichen. Dies kann sich wiederum auf die ersten beiden Schritte auswirken, sodass diese drei Schritte in der Regel mehrmals durchgearbeitet werden, bis die Typen in sich schlüssig sind

und bei einem weiteren Durchgang nicht mehr angepasst werden müssen.

Im letzten Schritt können die Typen abschließend charakterisiert und exemplarische Einzelfälle, die den jeweiligen Typus am treffendsten beschreiben, im Sinne eines Prototyps zur Verdeutlichung herausgegriffen werden. Entgegen dem z. B. von Gerhardt (2001) vorgeschlagenen Vorgehen, die ausgewählten Fälle in Bezug auf das der jeweiligen Kategorie Charakteristische zuzuspitzen, wird in dieser Studie der Einzelfall möglichst realitätsnah dargestellt und dabei seine Besonderheiten textlich herausgestrichen, um den jeweiligen Typus zu charakterisieren.

Abschließend sei erwähnt, dass die konstruierten Typen, die einem erwarteten Verhalten entsprechen, nicht moralisch oder objektiv wertend zu verstehen sind, wenngleich aufgrund ihrer Konstruktion eine *„subjektive Wertung unverkennbar dahinter steht“* (Janoska-Bendl 1965, S. 54).

6.8 Kontrolle der Datenanalyse

Aufgrund der Tatsache, dass der Forscher nicht in ein größeres Forschungsprojekt oder ein akademisches Setting eingebunden war, konnte die Kontrolle der Datenanalyse durch gemeinsames Codieren und gegenseitiges Prüfen nicht gewährleistet werden. Aus diesem Grund wurde die Strategie der detaillierten Darstellung des Analyseprozesses gewählt (Kvale 1996).

Zusätzlich wurde laufend der Austausch mit anderen Forschern gesucht. Bei mehreren Präsentationen an der TU Berlin konnte das methodische Vorgehen erläutert und diskutiert werden. Auch die regelmäßige Teilnahme am Forschungsseminar des „Chair of Strategic Management and Innovation“ an der ETH Zürich gab wertvolle Anregungen. Weiterer punktueller Austausch mit erfahrenen qualitativen Forschern aus anderen Themenbereichen waren ebenfalls hilfreich für die laufende kritische Reflexion des methodischen Vorgehens.

6.9 Grenzen des methodischen Vorgehens

Aufgrund der Tatsache, dass es sich bei der vorliegenden Studie um ein exploratives Vorgehen handelt, lässt sich schließen, dass der Leitfaden für die Interviews nicht optimal auf die

Dimensionen der Typenbildung ausgerichtet war, da sich diese erst im Verlaufe der Studie herauskristallisiert haben.

Es kann zudem als Schwäche bezeichnet werden, dass der Einfluss auf die Rekrutierung der Interviewpartner beschränkt war und somit nicht der ursprünglichen Technik des theoretischen Samplings (Glaser und Strauss 1967) – was durchaus erwünscht gewesen wäre – entsprach. Die gewählte Rekrutierung über die Firmen hat sicherlich auch zu einem anderen Sampling geführt, als wenn über die Community rekrutiert worden wäre, und somit dürften auch andere Ergebnisse zustande gekommen sein.

Wie bereits erwähnt, beanspruchte das gewählte methodische Vorgehen nicht, ein „*theoretisches System*“ (Kelle und Kluge 1999, S. 81) zu entwickeln. Somit können die Grenzen auch als Ausblick für Präzisierungen und Verbesserungen für nachfolgende Forschungsvorhaben verstanden werden.

Teil III: Ergebnispräsentation

7 Ergebnisse zu den Linux-Kernel-Logfiles

Alle im folgenden Text aufgeführten Tabellen sowie sämtliche Grundlagenwerte für die Grafiken und zahlreiche weitere tabellarische Auswertungen finden sich im Anhang F.

7.1 Umfang der Beiträge

Im Jahr 2007 wurden von mehr als 2.000 Entwicklern in über 28.000 Patches insgesamt etwa 1,7 Millionen Zeilen Code¹³⁰ hinzugefügt (siehe Tabelle 5).

	Entwicklung	Wartung
Autoren	2.097	374
Patches	28.334	1.335
Codezeilen	1.735.797	12.164
Personenjahre¹³¹	1.062	4
Monetarisierung in €¹³²	79.650.000	300.000

Tabelle 5: Übersicht der Beiträge am Linux-Kernel im Jahr 2007

Der Aufwand für die Wartung ist vergleichsweise gering. Da jedoch Fehler und Mängel in unkritischen Fällen durchaus nur im Entwicklungspfad oder direkt in den Distributionen behoben werden können, hat diese Information nur einen beschränkten Aussagewert über die Qualität der Entwicklung.

130 Diese Werte sind nur wenig niedriger als diejenigen von Jonathan Corbet in seinem Artikel unter <http://lwn.net/Articles/264440/> [10.08.2009]. Die vorhandene Differenz dürfte daran liegen, dass für die vorliegende Studie die Logfiles mit den härtesten Restriktionen erstellt wurden.

131 Für die Berechnung der Personenjahre wurde das weitverbreitete, wenn auch etwas vereinfachende COCOMO-Modell (Boehm 2000) mit den Standardwerten für mittelkomplexe Software verwendet. Die Formel dafür lautet: **Personenjahre = 3 * Tausend Codezeilen¹³² / 12**. Das aktuellere COCOMO-II-Modell konnte nicht verwendet werden, da dieses von logischen Codezeilen ausgeht, welche hier nicht zur Verfügung standen. Die Verwendung der physischen Codezeilen für Metriken hat zwar Mängel. Es gibt jedoch Grund zur Annahme, dass die Anzahl der geschriebenen Codezeilen mit der Leistung durchaus korreliert und somit verwendet werden kann.

132 Für die Ermittlung des monetären Wertes wurde die Basis der MERIT-Studie (2006, S. 49) hinzugezogen, welche als Jahreskosten für ein Personenjahr € 75.000 verwendet, was als vorsichtig bezeichnet wird. Die Monetarisierung wird somit auf der Basis von europäischen Werten vorgenommen, was bei der weltweiten Beteiligung nur einen informativen Wert hat.

7.2 Interessengruppen

Die Beiträge lassen sich nach der Motivation aus kommerziellen, öffentlichen¹³³ oder privaten Interessen kategorisieren. Abbildung 14 zeigt die prozentualen Anteile der vier Kategorien.

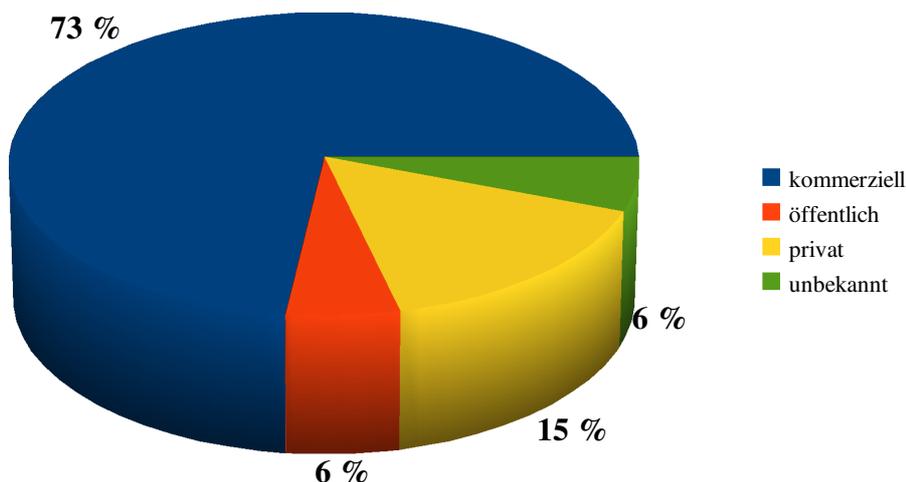


Abbildung 14: Motivation zur Mitarbeit

Beinahe 75 % der Entwicklung stammen von kommerziell orientierten Firmen. Zählt man die 6 % der öffentlichen Institutionen wie der Linux Foundation, Universitäten oder Regierungen hinzu, so werden rund 80 % der Codezeilen in einem bezahlten Arbeitsverhältnis geschrieben. Aus privater Initiative werden 20 % der Beiträge (einschließlich der Unbekannten¹³⁴) unentgeltlich geschrieben. Bei der Wartung ergibt sich nur eine leichte Verschiebung zugunsten der kommerziellen und öffentlichen Institutionen auf Kosten der Privaten und Unbekannten.

Berechnet man, wie unter 7.3.1 erläutert, für die vier Interessenbereiche die Personenjahre und die Monetarisierung, so lassen sich die kommerziell orientierten Aufwendungen mit 56 Mio. € monetarisieren (siehe Tabelle 6).

¹³³ Zu den öffentlichen Institutionen zählen neben den Regierungen und Universitäten auch Non-Profit-Organisationen wie z. B. die Linux Foundation. Diese Unterscheidung nach kommerziellen und öffentlichen Institutionen wird in den Statistiken vom lwn.net so nicht gemacht. Dort werden beide zusammen unter „employer“ geführt, was natürlich eine etwas andere Aussage ist.

¹³⁴ Wie aus den Detailauswertungen ersichtlich ist, gibt es guten Grund zur Annahme, dass es sich bei den nicht bekannten Autoren tatsächlich zumeist um Privatpersonen handelt.

	Personenjahre	Monetarisierung in €
kommerziell	748	56.099.027
öffentlich	42	3.145.749
privat	132	9.872.124
unbekannt	44	3.281.742

Tabelle 6: Personenjahre und Monetarisierung nach Interessengruppen

Interessant in diesem Zusammenhang ist, dass ein Entwickler im kommerziellen Umfeld nach COCOMO im Durchschnitt 3,3 Monate, im öffentlichen Sektor 1,8 Monate und Private 1,9 Monate Zeit (bei den Unbekannten sind es 0,7) aufwenden.

7.3 Beteiligte Firmen

7.3.1 Top-Firmen

Im Jahr 2007 leisteten Mitarbeitende von insgesamt 367 Firmen Beiträge in unterschiedlichem Umfang. Die Top-10-Firmen sind in Tabelle 7 aufgelistet:

	Firma	Codezeilen		Patches		Autoren	
1	Red Hat	185.518	10,69 %	2.741	9,67 %	90	4,29 %
2	Intel	126.062	7,26 %	923	3,26 %	65	3,10 %
3	IBM	123.173	7,10 %	2.293	8,09 %	144	6,87 %
4	Analog Devices	101.074	5,82 %	357	1,26 %	11	0,52 %
5	Novell	91.195	5,25 %	2.161	7,63 %	56	2,67 %
6	SGI	44.646	2,57 %	1.074	3,79 %	30	1,43 %
7	Freescal	31.772	1,83 %	427	1,51 %	30	1,43 %
8	linutronix	31.215	1,80 %	506	1,79 %	5	0,24 %
9	Oracle	27.229	1,57 %	672	2,37 %	19	0,91 %
10	Renesas Technology	22.300	1,28 %	397	1,40 %	5	0,24 %

Tabelle 7: Beiträge von den Top-10-Firmen

Die drei Firmen Red Hat (10.7 %), Intel (7.3 %) und IBM (7.1 %) trugen dabei ein gutes Viertel des gesamten Codes bei. Zählt man noch Analog Devices (5.8 %) und Novell (5.2 %) hinzu, so ergibt dies bereits deutlich mehr als ein Drittel an Code, der alleine durch fünf Unternehmen beigetragen wurde. Als erster reiner Dienstleister liegt das deutsche Unternehmen

Linutronix auf Platz 8 (1.8 %), als erster echter Softwareanbieter Oracle auf Platz 9 (1.5 %).

7.3.2 Firmen nach Regionen

Die geografische Verteilung der beitragenden Firmen weist eine klare Dominanz der westlichen Welt auf. Zu beachten ist dabei jedoch, dass es sich nicht um das jeweilige Domizil des Autors handelt, sondern um den Hauptsitz der Firma.¹³⁵ Nach Kontinenten aufgeteilt präsentiert sich die Abbildung 15:

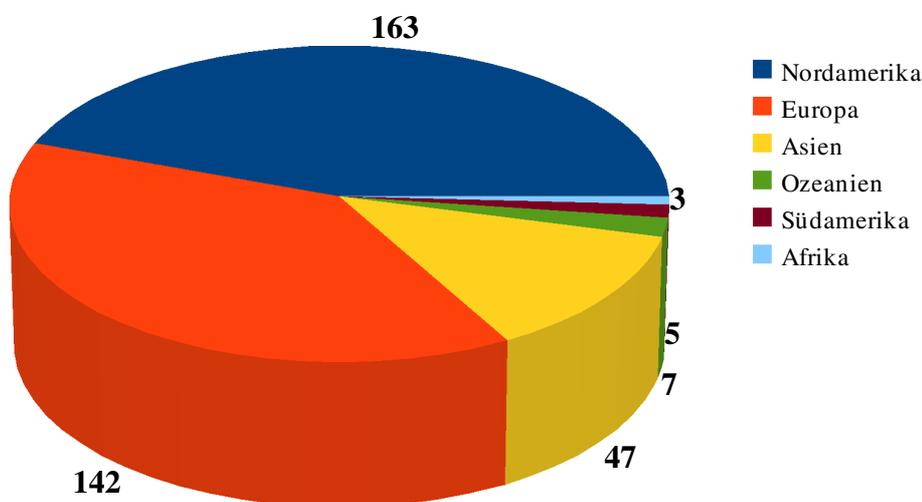


Abbildung 15: Anzahl der Firmen, aufgeteilt nach Kontinenten

Aufgrund dieser Zahlen zeigt sich ein digitaler Graben. Der gesamte südamerikanische Raum ist mit nur drei beitragenden Firmen in Brasilien sowie zwei in Argentinien nahezu inexistent. Dieses Ergebnis wird verdeutlicht durch die Anzahl beigetragener Codezeilen, die lediglich im Promillebereich liegen. In Afrika ist nur Südafrika präsent, dies jedoch in Bezug auf die geleisteten Codezeilen in einer weitaus größeren Dimension (0.53 %), was in erster Linie auf die Präsenz des Ubuntu-Distributoren Canonical zurückzuführen ist.

Der asiatische Raum ist weitaus stärker beteiligt. Japan ist in Bezug auf beigetragene Codezeilen hinter den USA gar das zweitgrößte Land (6.01 %). Taiwan (0.57 %) liegt in der Be-

¹³⁵ Diese Betrachtung ist sicherlich nicht ideal. Aufgrund des vorhandenen Materials war jedoch eine genauere Lokalisierung nicht möglich. Die Aussagekraft liegt somit in erster Linie in der strategischen Bedeutung von Linux in den Firmen und weniger in der dezentralen Verteilung der Entwickler.

deutung für den Linux-Kernel noch weit vor Israel (0.27 %), Korea (0.11 %), Indien (0.07 %) und Singapur (0.05 %), dem ebenfalls dem asiatischen Raum zugeteilten Russland (0.01 %) und China (0.00 %).

Die USA sind mit 145 Firmen (39.51 %) und beträchtlichen 80 % der Codezeilen dominierend. Mit AT&T, Hewlett-Packard, IBM, Dell, Intel, Motorola, Cisco, Comcast, General Dynamics, Oracle, Google, Sun, Texas Instruments, EMC, AMD, Micron, Unisys und Rockwell Automation sind 18 der US-amerikanischen Fortune-500-Firmen¹³⁶ dabei, von denen immerhin neun gar in den Top 100 zu finden sind. Auch Kanada leistet mit 1.32 % der Codezeilen durch insgesamt 18 Firmen (4.90 %) einen bedeutenden Beitrag.

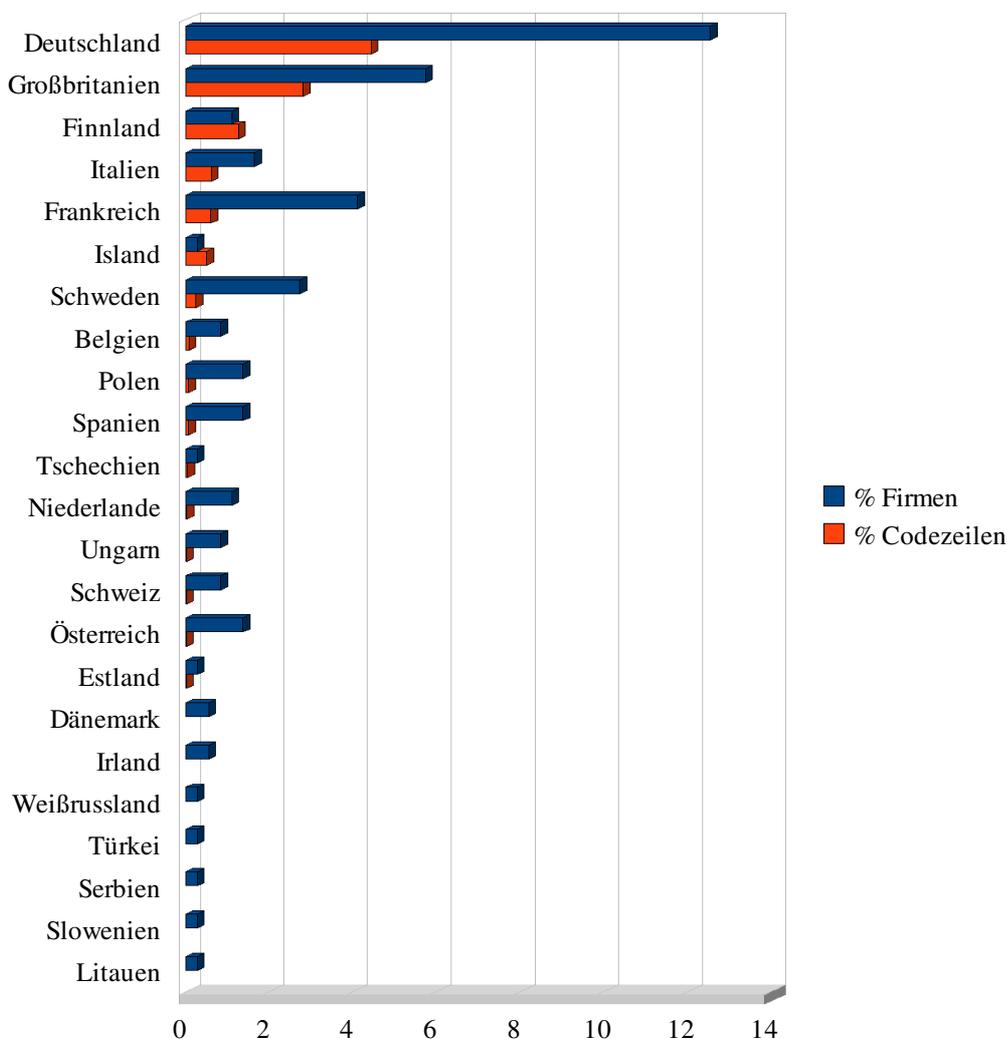


Abbildung 16: Anzahl Firmen und Codezeilen aus Europa, aufgeteilt nach Ländern

136 Siehe http://money.cnn.com/magazines/fortune/fortune500/2008/full_list/index.html [10.08.2009].

In Europa ist Deutschland am stärksten engagiert. Dass Finnland an prominenter Stelle erscheint, liegt in erster Linie daran, dass das finnische Unternehmen Nokia im Kernel aktiv ist. Ansonsten dominieren eher die industriellen „Schwergewichte“ bezüglich der Anzahl der beitragenden Firmen (siehe Abb. 16).

7.3.3 Firmen nach Größe

Die Firmen wurden in fünf Kategorien von sehr klein bis sehr groß eingeteilt. Die Einteilung wurde weitgehend vom frei zugänglichen Teil der Amadeus-Datenbank übernommen, obwohl dort nicht ganz klar ersichtlich ist, wie sich die Firmengröße genau definiert.¹³⁷ Dennoch lassen sich einige interessante Schlüsse ziehen.

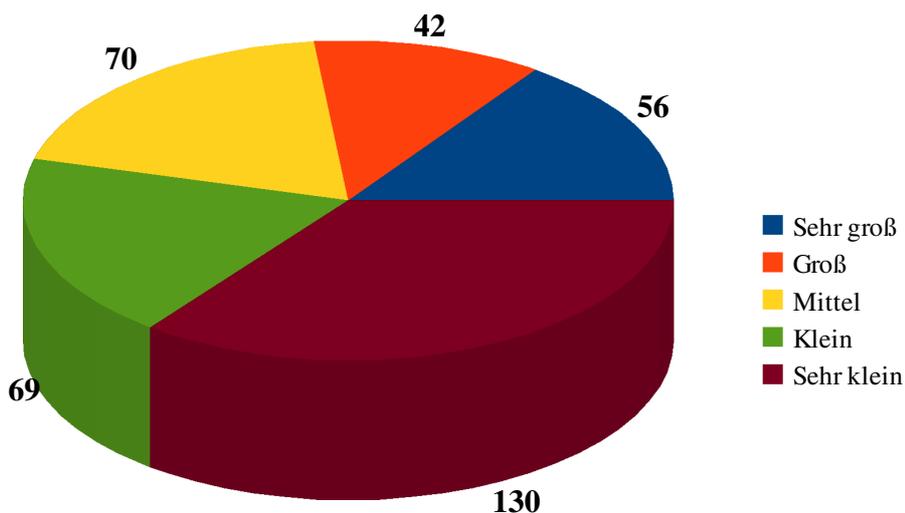


Abbildung 17: Anzahl der Firmen, nach Größe gruppiert

Der Anteil der kleinen und sehr kleinen Firmen liegt über 50 % (siehe Abb. 17). Da die meisten sehr großen Firmen in den USA bzw. in Asien domiziliert sind, ist der prozentuale Anteil der kleineren Firmen in Europa sogar noch etwas größer.

Betrachtet man hingegen die Anzahl der Linux-Kernel-Entwickler sowie die 2007 geänderten Codezeilen, so haben die sehr großen (42.94 % bzw. 48.17 %) und großen Firmen (22.08 % bzw. 30.39 %) doch eine viel größere Bedeutung.

¹³⁷ Auf jeden Fall handelt es sich um einen Mix aus unterschiedlichen Werten und nicht bloß aus der Anzahl der Mitarbeiter oder dem Umsatz.

7.3.4 Firmen nach Sektoren

Wie bereits in der Beschreibung des Vorgehens (siehe Kapitel 5.4) erwähnt, ist die Einteilung der Firmen in Sektoren aufgrund des vorhandenen Datenmaterials sehr schwierig. Ein zusätzliches Problem ist, dass sich insbesondere die großen, klassischen Hardwarehersteller wie IBM, HP oder Sun durch einen Mix aus Hardware, Software und Dienstleistungen (umsatzmäßig etwa zu je einem Drittel) auszeichnen, jedoch nur eine Zuteilung vorhanden war (bei IBM z. B. Software & Dienstleistungen, bei HP und Sun Hardware). Unter diesen Einschränkungen ergibt sich folgendes Bild:

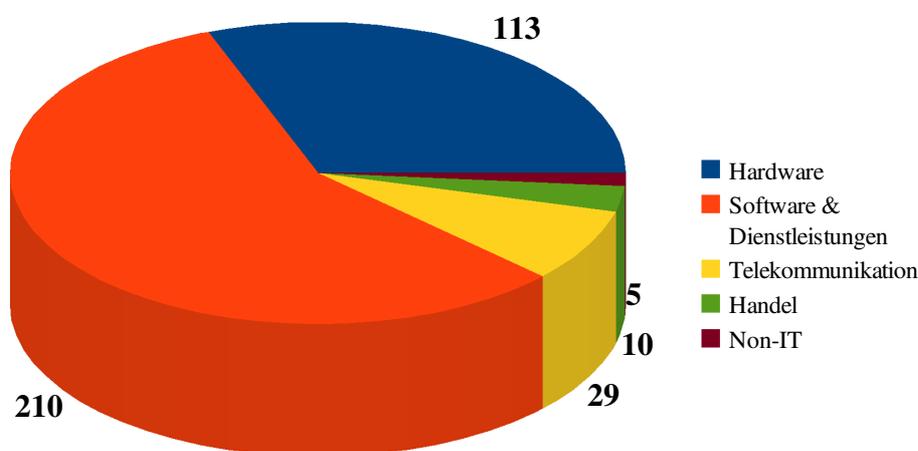


Abbildung 18: Anzahl Firmen nach Sektoren gruppiert

Wie Abbildung 18 aufzeigt, haben Firmen aus dem Bereich Software und Dienstleistungen den größten Anteil (57.22 %), deutlich mehr als die Hardwarevendors (30.79 %). Der Handel und Non-IT-Firmen – insbesondere der Anteil der beigetragenen Codezeilen (0.44 % bzw. 0.10 %) – weisen einen deutlich geringeren Anteil auf.

7.3.5 Firmen nach Anzahl der Linux-Entwickler

Betrachtet man die Firmen nach der Anzahl der Autoren, die in ihrem Namen im Jahre 2007 einen oder mehrere Patches zum Linux-Kernel beigetragen haben, lässt dies eine klare Aussage zu, wie Abbildung 19 aufzeigt.

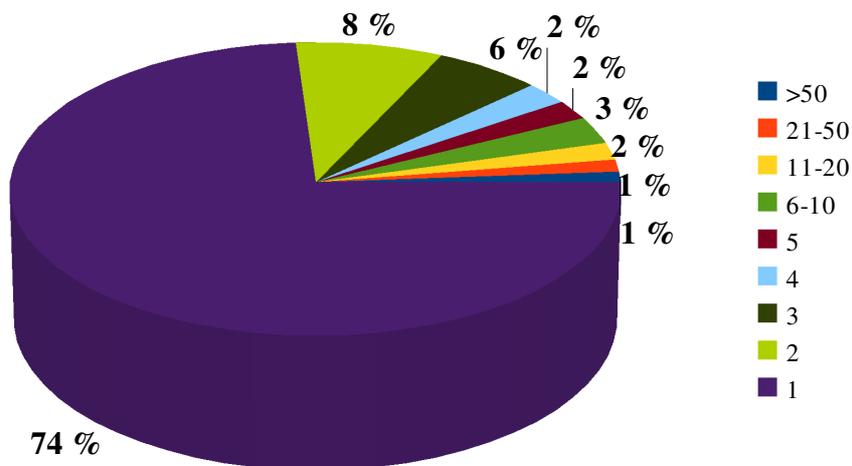


Abbildung 19: Gruppierung der Firmen nach Anzahl der Autoren

Rund drei Viertel der insgesamt 367 am Linux-Kernel im Jahre 2007 beteiligten Firmen ließen einen einzigen Entwickler am Linux-Kernel mitarbeiten, während 16 Firmen (4.5 %) mehr als zehn Entwickler einsetzten.

Die Frage stellt sich nun, wie sich die Beiträge auf diese Firmen verteilen.

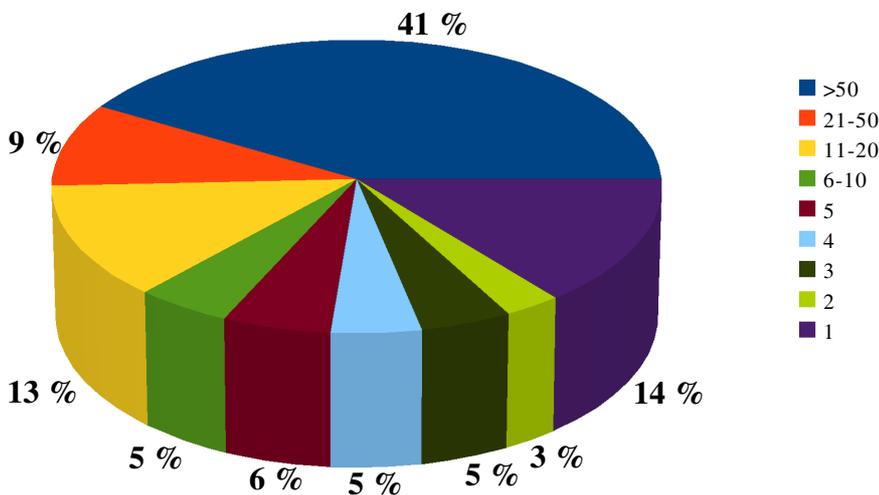


Abbildung 20: Beiträge der Firmen, gruppiert nach Anzahl der Autoren pro Firma

Abbildung 20 zeigt auf, dass die 16 Firmen mit mehr als zehn Linux-Entwicklern insgesamt 63 % der von kommerziell orientierten Unternehmen beigetragenen Codezeilen beisteuern (was einem Anteil von mehr als 45 % aller Beiträge entspricht). Dies zeigt einerseits, dass große Firmen einen beträchtlichen Teil am Linux-Kernel beitragen (was schon die Summe der

Beiträge der Top-5-Firmen klar aufgezeigt hat), andererseits verteilen sich aber mehr als die Hälfte der Codebeiträge auf kleinere Firmen, öffentliche Institutionen sowie Privatpersonen. Zudem stellen nur gerade vier Firmen (Red Hat, Novell, Intel und Linutronix) mehr als einen Entwickler aus den Top 30.

Mit nur zwei Ausnahmen (MontaVista und SWsoft) sind alle Firmen, die zehn oder mehr Entwickler stellen, große oder sehr große Unternehmen. Mit Ausnahme von zwei japanischen Konzernen sind zudem alle in den USA domiziliert. Mehr als die Hälfte dieser Firmen sind dem Hardwaresektor zuzurechnen. Allerdings gibt es einige kleinere Firmen, die bis zu fünf Entwickler am Linux-Kernel mitarbeiten lassen. Umgekehrt gibt es relativ wenige sehr große Firmen (19 von 59), die nur einen Entwickler engagieren.

7.4 Kategorisierung der Firmenbeiträge

Der Linux-Kernel ist in folgende Module unterteilt (Kroah-Hartman 2007): eigentlicher Kern („core“), Treiber („drivers“), Architekturen („architecture“), Netzwerk („network“), Dateisystem („filesystems“) sowie Diverses („miscellaneous“).¹³⁸

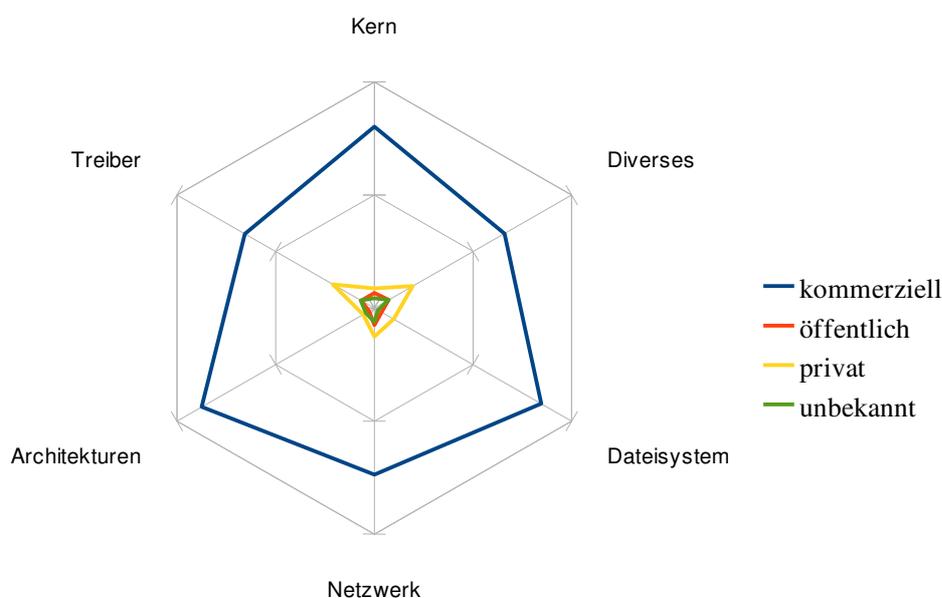


Abbildung 21: Codebeiträge zu den Modulen nach Interessengruppen

¹³⁸ Diese Einteilung entspricht leider nicht ganz der Verzeichnisstruktur, sodass die Zuteilung manuell gemacht werden musste.

Wie der Abbildung 21 zu entnehmen ist, leisten in jedem Modul die kommerziellen Firmen den Hauptbeitrag, jedoch nicht überall im selben Ausmaß (die Achsen zeigen jeweils 100 % der Codebeiträge eines Moduls). Betrachtet man die Aufteilung auf die Module nicht hinsichtlich der absoluten Anzahl der beigetragenen Codezeilen, sondern jeweils in Bezug zur Gesamtheit der eigenen Beiträge, so ergibt sich ein sehr interessantes Bild.

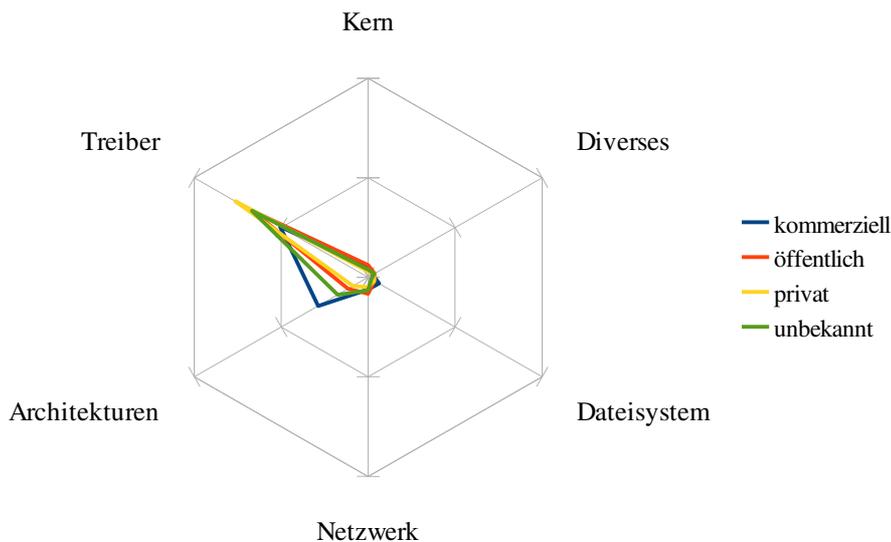


Abbildung 22: Codebeiträge zu den Modulen in Bezug zur Gesamtheit der Interessengruppe

Die Interessen sehen wie in Abbildung 22 ersichtlich unabhängig von der kommerziellen Ausrichtung recht ähnlich aus. Die Firmen engagieren sich auf Kosten der Treiber mehr für die Architektur. Aber weder die Privaten noch die öffentlichen Institutionen richten ihr Augenmerk deutlich auf eines der anderen Module.

Interessant ist, dass das Filesystem fast ausschließlich durch US-amerikanische Firmen entwickelt wird (97.93 %). In Europa sieht das Bild so aus, dass der Architekturbereich unter (6.62 %), dafür der Treiber- sowie der Netzwerkbereich überproportional vertreten sind (13.19 % bzw. 15.21 %). In Asien ist das Verhältnis hingegen umgekehrt (11.8 % gegenüber 6.12 % bzw. 2.78 %).

Zudem unterscheiden sich die sehr großen von den großen Firmen in ihrer Bedeutung in Bezug auf die Kategorien, wie aus Abbildung 23 ersichtlich ist.

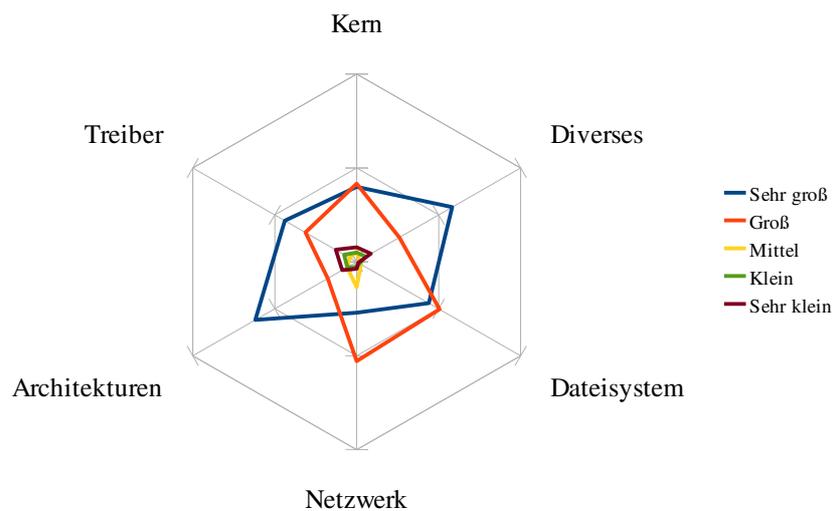


Abbildung 23: Codebeiträge zu den Modulen nach Firmengröße

Während die sehr großen Firmen den Architekturbereich sowie das Diverse mit 61.84 % bzw. 58.26 % dominieren, sind die „nur“ großen Firmen im Netzwerkbereich (52.82 %) sowie bei den Filesystemen (50.52 %) für über die Hälfte der Codezeilen verantwortlich. Dies hat vor allem damit zu tun, dass die Firma Red Hat, die beim Netzwerk (20.1 %) und Filesystem (23.25 %) mit Abstand führend ist, lediglich als große Firma eingestuft ist. Kleine und sehr kleine Firmen konzentrieren sich stark auf die Entwicklung von Treibern. 62.49 % bzw. 65.16 % der gesamten beigetragenen Codezeilen von Firmen dieser Größe gehören dazu.

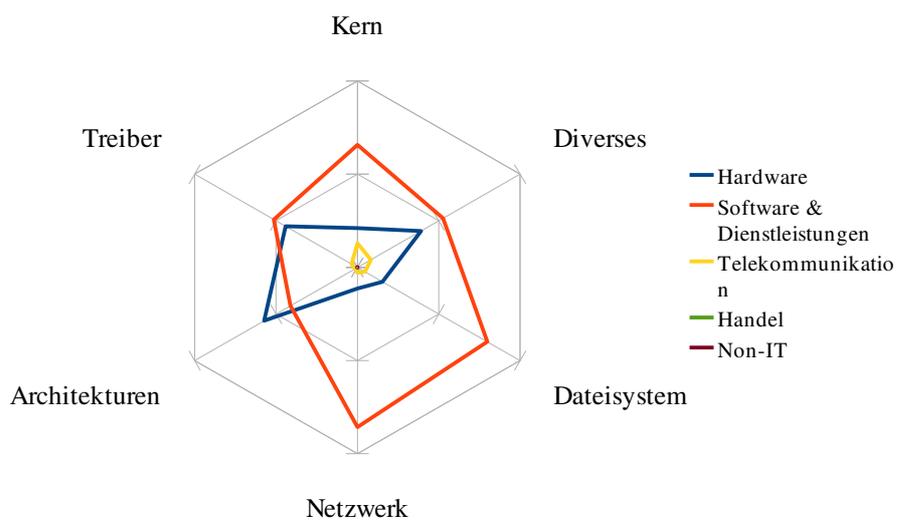


Abbildung 24: Codebeiträge zu den Modulen nach Sektoren

Die Aufteilung der Kategorien nach Sektoren wird in der Abbildung 24 dargestellt. Mit Ausnahme der hardwarenahen Bereiche Treiber und Architektur sind vor allem Softwarehäuser und Dienstleister in der Entwicklung engagiert. Überraschend ist jedoch, dass die Telekommunikations-Firmen nicht mehr zum Netzwerkbereich beitragen und wie alle anderen Sektoren hauptsächlich in die Entwicklung von Treiber investieren (siehe Abb. 25).

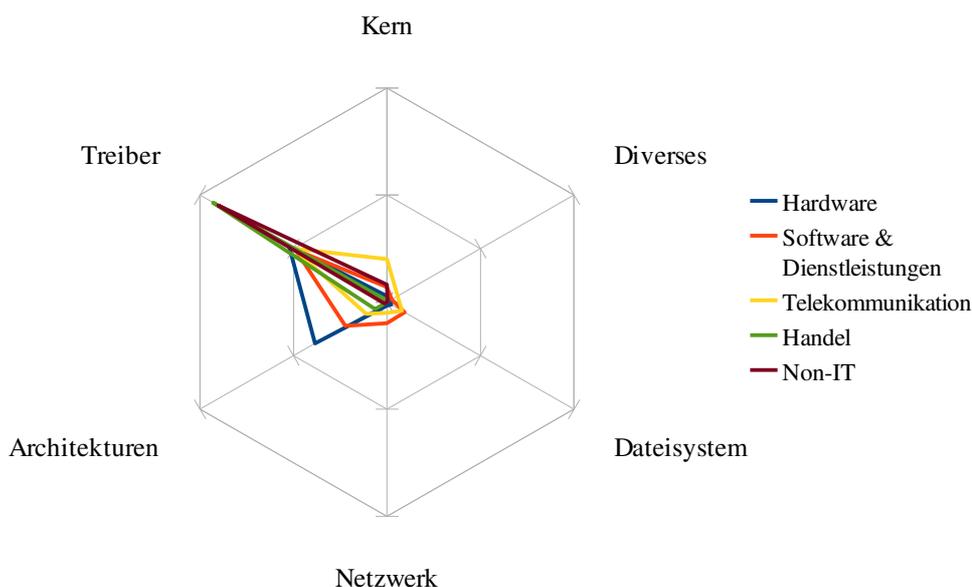


Abbildung 25: Codebeiträge zu den Modulen nach Sektoren in Bezug zur Gesamtheit des Sektors

7.5 Aufteilung nach Architekturen

Linux wurde zwar ursprünglich durch Linus Torvalds programmiert, um ein UNIX-kompatibles Betriebssystem auf seinem billigen 386er-Intel-PC laufen lassen zu können (Torvalds 2001). Es wurde jedoch in der Zwischenzeit auf viele Hardwareplattformen portiert. Der Kernel ist so aufgebaut, dass dabei die verschiedenen Plattformen als selbstständige Module funktionieren und so relativ unabhängig betreut werden können. Wie Abbildung 26 aufzeigt und wie schon anhand der Aufteilung nach Kategorien ersichtlich war, sind die Firmen proportional stärker an der Unterstützung der Architekturen beteiligt:¹³⁹

¹³⁹ Zu den Architekturen wurden nicht nur die Kategorie „architecture“ gezählt, sondern es sind auch noch klar

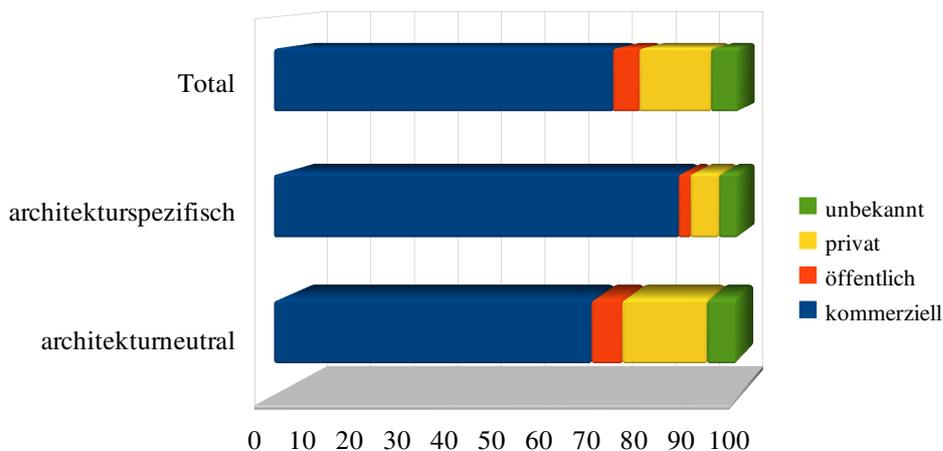


Abbildung 26: Beteiligung an der Architektur im Total pro Interessengruppe

Die Unterstützung der einzelnen Architekturen stellt sich jedoch nicht homogen dar, wie Abbildung 27 aufzeigt:

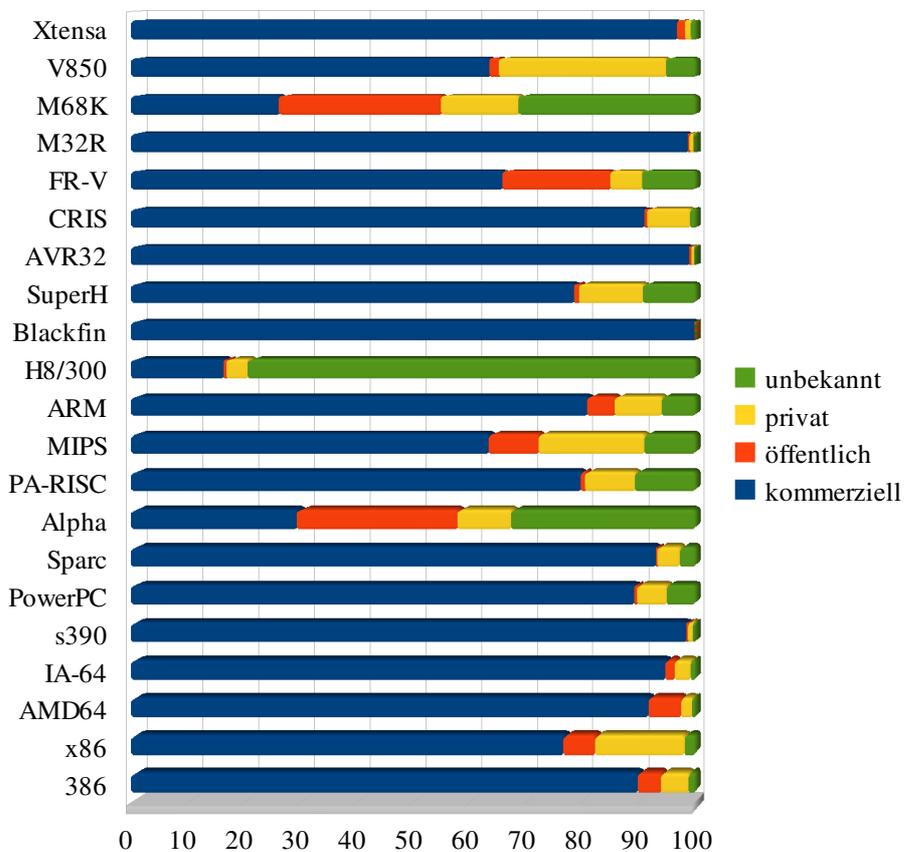


Abbildung 27: Architekturen nach Interessengruppe in %

zuteilbare Quellen aus anderen Kategorien hinzugekommen, insbesondere aus der Kategorie der Treiber.

Dass die Architekturen, die für High-end Server stehen, vor allem durch die Firmen unterstützt werden, ist nahe liegend. Ersichtlich ist jedoch auch, dass Architekturen, die man im weitesten Sinne dem Consumer-Bereich zuteilen kann, viel stärker nicht kommerziell Interessierte anzieht. Erstaunlich ist hingegen, dass die 386er-Architektur nicht stärker durch Private betreut wird. Der hohe Anteil von Unbekannten bei der Architektur H8/300 lässt darauf schließen, dass einige von ihnen zur Herstellerfirma Hitachi gehören, was in geringerem Maße auch bei M68K (Motorola) und Alpha (HP) der Fall sein dürfte.

Die Architekturen werden, wie bereits gezeigt, in der Regel überdurchschnittlich durch kommerziell orientierte Firmen bearbeitet. Die Frage ist nun, ob dies ausschließlich Herstellerfirmen sind, oder ob auch andere Firmen – wie etwa Distributoren – mitarbeiten.

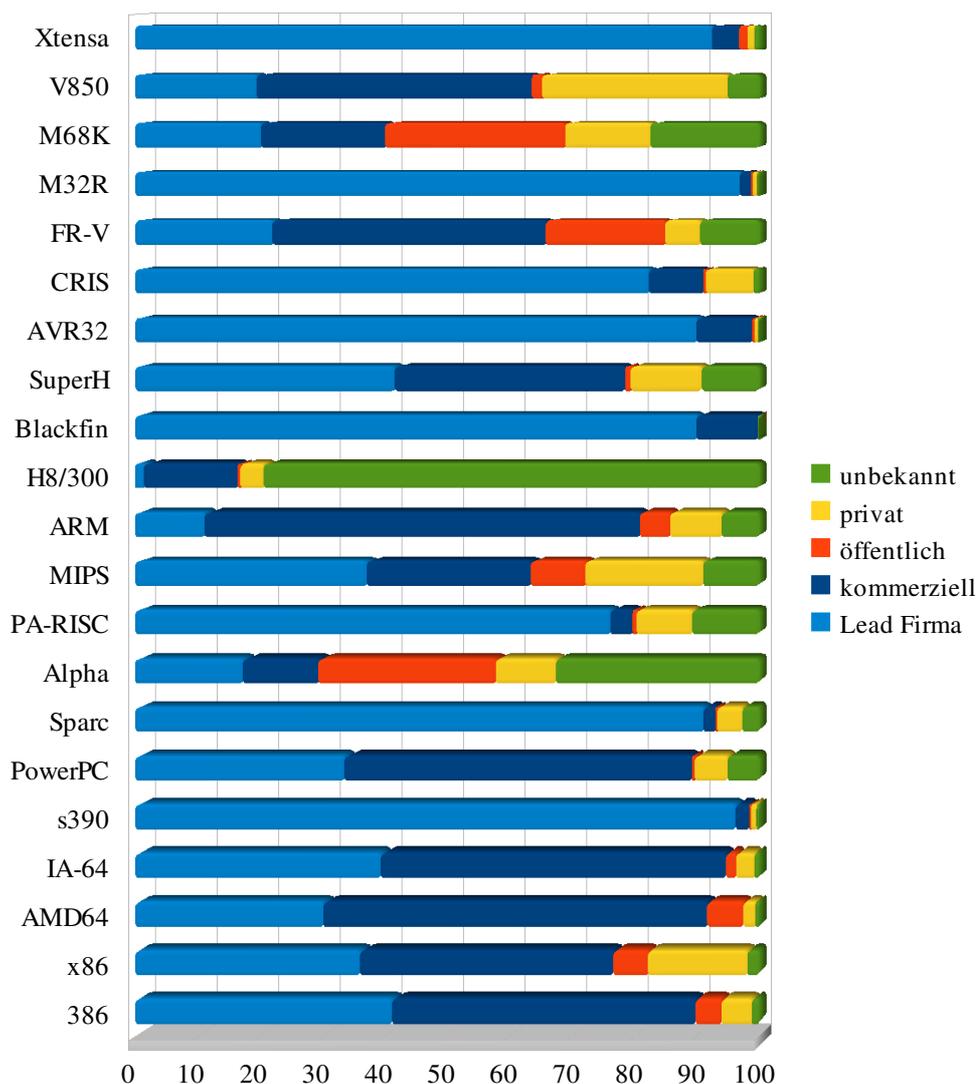


Abbildung 28: Architekturen nach Herkunft in % mit Lead Firma

Weniger als die Hälfte der Architekturen wird durch eine Firma klar dominiert (siehe Abb. 28). In der Regel ist die dominierende Firma der Hersteller, in zwei Fällen allerdings auch der Distributor Red Hat. Auch wird deutlich, dass die High-End-Architekturen in besonderem Ausmaß von den Herstellern unterstützt werden.

Beteiligt an der Architektur waren im Jahr 2007 nur 144 der insgesamt 367 Firmen (39 %). Die Top-30-Firmen tragen insgesamt über 80 % des Codes bei, die restlichen 114 nur noch 7 %. Betrachtet man, wer in absoluten Zahlen am meisten Codezeilen zu den Architekturen beiträgt, so ergibt sich ein überraschendes Bild: Analog Devices führt die Liste mit fast doppelt so viel beigetragenen Codezeilen (21 %) deutlich vor IBM (13 %) an. Es folgen Red Hat (7 %), SGI (5 %) und Freescale (5 %). Überraschend sind auch die auf der Liste fehlenden Firmen: zum einen HP (0.3 %) und Sun (0.03 %), die die Unterstützung ihrer Hardware offensichtlich weitgehend durch den Distributor Red Hat gewährleisten lassen. Andererseits hält sich die Linux Foundation (0.09 %) zurück. Noch interessanter in diesem Zusammenhang ist, wie stark sich die Firmen auf die Hardwareunterstützung konzentrieren. Während Analog Devices (91 % ihrer Beiträge gehen an die Architektur Blackfin), Tensilica (100 % Xtensa) und Axis Communications (99 % CRIS) sich fast ausschließlich auf ihre eigene Architektur konzentrieren, sind IBM (nur 45 % ihrer Codebeiträge sind direkt den Architekturen zuzuschreiben) und SGI (48 %) doch etwas breiter integriert.

8 Linux-Kernel-Entwickler in großen IKT-Firmen

8.1 Soziodemografischer Überblick

Der soziodemografische Überblick bezieht sich auf die befragten Linux-Kernel-Entwickler. Es wurden sieben weitere Personen innerhalb der Firmen befragt, die selbst nicht (mehr) entwickeln und entweder Projekt- oder Produktmanager sind und damit als Open-Source-Evangelisten bezeichnet werden können.

Insgesamt nahmen siebzehn Entwickler in fünf Firmen an den Interviews teil. Drei der Firmen sind in Deutschland, je eine in der Schweiz und in den USA domiziliert. Alle Firmen haben eine amerikanische Muttergesellschaft.¹⁴⁰ Die unterschiedliche Anzahl der Befragten pro Firma kam einerseits dadurch zustande, dass die Firmen in unterschiedlicher Zahl Linux-Kernel-Entwickler beschäftigen. Andererseits wurde der Zugang nicht überall gleich offen gewährt oder es haben sich nur wenige Entwickler zu einem Interview bereit erklärt.

Es konnten Interviews mit Entwicklern beider Geschlechter geführt werden, die Verteilung ist allerdings sehr einseitig auf Männer ausgerichtet. Diese Geschlechterverteilung entspricht ungefähr derjenigen in der deutschen Softwarebranche.¹⁴¹ Die Interviewteilnehmenden sind in ihrer Arbeit durchwegs sehr technisch orientiert, eine Managementkarriere wird nicht angestrebt. Dies ist vermutlich dem hohen, meist technikorientierten Ausbildungsstand zuzuschreiben und weniger dem speziellen Untersuchungsfeld.

Sehr viele Linux-Kernel-Entwickler werden direkt aus den Hochschulen rekrutiert. Der Wechsel intern von einem Closed-Source-Projekt zu Linux ist danach ein beliebter Weg. Eher selten werden Entwickler über die Community angefragt. In der Regel bewirbt sich ein Interessent auf eine Stellenausschreibung, sowohl bei einer externen als auch bei einer internen Neubesetzung. Eine direkte Anfrage an die Community seitens der Firma ist auch bei internen Wechseln eine Ausnahme.

¹⁴⁰ Genauere Angaben zu den Firmen können leider nicht gemacht werden, da ansonsten die Anonymität nicht mehr gewährleistet werden kann.

¹⁴¹ Der Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V. (BITKOM) schätzt, dass ca. 30.000 bis 50.000 Frauen im Bereich Informatik tätig sind (BITKOM 2007a; 2007b), was bei insgesamt rund 830.000 Beschäftigten (BITKOM 2009) 3,5–6 % ausmacht.

Die Dauer der Firmenzugehörigkeit der Interviewteilnehmenden reicht von ein paar Monaten bis zu 21 Jahren. Entsprechend dem Alter divergiert auch die Erfahrung in der Softwareentwicklung stark. Die meisten Hochschulabschlüsse waren zumindest der Informatik nahe. Nur eine Person absolvierte den Abschluss auf dem zweiten Bildungsweg.

Code	Firma	Geschlecht	Alter ¹⁴²	Ausbildung ¹⁴³	Rekrutierung
B2	A	m	37	Hochschule	intern
C3	A	m	30	Fachhochschule	extern
D4	A	m	33	Fachhochschule	intern
E5	A	w	44	Hochschulabschluss	intern
F6	A	m	42	Promoviert	intern
G7	A	m	37	Promoviert	direkt aus der Hochschule
H8	B	m	32	Hochschulabschluss	direkt aus der Hochschule
I9	B	m	29	Fachhochschule	direkt aus der Hochschule
J10	B	m	37	Promoviert	von Community
K11	B	m	36	Hochschulabschluss	extern
L12	B	m	39	Promoviert	direkt aus der Hochschule
N14	C	m	28	Hochschulabschluss	direkt aus der Hochschule
O15	D	m	43	Promoviert	extern
R18	B	m	36	Lehre	von Community
S19	E	m	35	Hochschulabschluss	extern
T20	E	m	35	Hochschulabschluss	intern
U21	E	m	28	Hochschulabschluss	direkt aus der Hochschule

Tabelle 8: Soziodemografischer Überblick der interviewten Linux-Kernel-Entwickler

Tabelle 8 bietet einen Überblick der soziodemografischen Daten der Interviewten Linux-Kernel-Entwickler.

142 Zum durchschnittlichen Alter der IT-Beschäftigten in Deutschland konnten keine verlässlichen Vergleichsdaten gefunden werden. Aufgrund der Tatsache, dass das Aufkommen von Linux vor allem von den heute 31- bis 40-Jährigen aktiv miterlebt werden konnte, ist anzunehmen, dass diese Altersgruppe überdurchschnittlich repräsentiert ist.

143 Zum Ausbildungsstand der IT-Beschäftigten konnten keine verlässlichen Daten gefunden werden. Anhand der Anforderungen an die offenen Stellen kann jedoch vermutet werden, dass die in der Studie vorgefundenen Werte durchaus vergleichbar sind (BITKOM 2007b).

8.2 Die empirisch begründete Typenbildung

Die zentralen Vergleichsdimensionen, die sich aus der Datenanalyse ergeben haben, sind die Motivation sowie das Werte- und Normensystem der Entwickler. Es handelt sich hierbei um zwei Dimensionen, die bereits sehr früh in der Diskussion um Open-Source-Software anzutreffen sind (siehe dazu Kapitel 3.2).

Die Dimension Motivation ist dabei durch folgende zwei Ausprägungen charakterisiert:

Extrinsische Motivation: Programmierer mit dieser Ausprägung sind hauptsächlich durch monetäre Anreize, gute Berufsaussichten sowie einen konkreten Bedarf in ihrer Tätigkeit motiviert.

Intrinsische Motivation: Programmierer mit dieser Ausprägung sind hauptsächlich durch den Spaß am Programmieren, das Streben nach Reputation, das Zugehörigkeitsgefühl, das Lernbedürfnis sowie durch Altruismus in ihrer Tätigkeit motiviert.

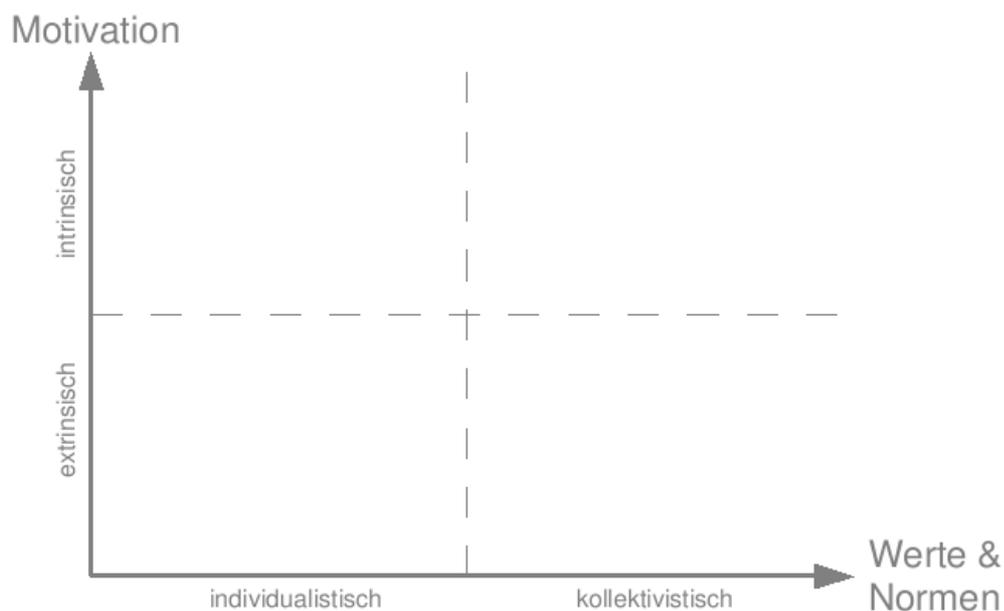


Abbildung 29: Dimensionen für die Typenbildung

Die Dimension des Normen- und Wertesystems zeichnet sich durch die folgenden zwei Ausprägungen aus (siehe Abb. 29):

Sich am Individuellen orientierende Normen und Werte: Programmierer dieser Ausprägung

orientieren sich hauptsächlich an den Normen und Werten einer Person, einer Gruppe bzw. einer Organisation.

Sich am Kollektiven orientierende Normen und Werte: Programmierer dieser Ausprägung orientieren sich hauptsächlich an den Normen und Werten des ganzen Kollektivs.

Aufgrund der Dimensionen und deren Ausprägungen konnten folgende Typen aus den Daten eruiert werden:

- Der *Pragmatische Ingenieur*, der extrinsisch motiviert ist und sich an den Normen und Werten des Individuellen orientiert.
- Der *Dialektische¹⁴⁴ Informatiker*, der ebenfalls extrinsisch motiviert ist, sich jedoch an den Normen und Werten des Kollektiven orientiert.
- Der *Sozialromantische Hacker*, der intrinsisch motiviert ist und sich an den Normen und Werten des Kollektiven orientiert.

Ein möglicher vierter Typ, der intrinsisch motivierte, sich an den Normen und Werten des Individuellen orientierende Programmierer, den man im weitesten Sinne als Hedonisten bezeichnen könnte, war in den Daten nicht als Typ erkennbar. Allerdings lassen sich entsprechende Ausprägungen durchaus finden, sie werden deshalb ergänzend in der Ergebnispräsentation aufgezeigt.

Die Verteilung auf die drei Typen stellt sich wie in Tabelle 9 aufgelistet dar:

144 Zur Begrifflichkeit der Dialektik siehe z. B. Popper (1993).

Der pragmatische Ingenieur	Der dialektische Informatiker	Der sozialromantische Hacker
B2	C3	R18
D4	G7	
E5	H8	
F6	J10	
I9	U21	
K11		
L12		
N14		
O15		
S19		
T20		

Tabelle 9: Verteilung der Interviewteilnehmenden auf die drei Typen

Es wird ersichtlich, dass die Gewichtung der Typen sehr unterschiedlich ist. Während es mehr pragmatische Programmierer gibt, ist der sozialromantische Programmierer nur ein einziges Mal anzutreffen.

8.3 Der pragmatische Ingenieur

8.3.1 Charakterisierung des Prototyps: Franz¹⁴⁵ (F6)

8.3.1.1 Werdegang

Franz ist ein 42 Jahre alter Familienvater. Er studierte im Hauptfach Informatik. Sein Nebenfach Chemie wählte er bewusst so, dass er eine aus seiner Sicht seltene und in Bezug auf Berufsaussichten zukunftssträchtige Kombination vorweisen kann. Nach seiner Doktorarbeit über ein hardwarenahes Thema, die er während seiner Tätigkeit als Assistent mit Lehrverpflichtung an einer Universität geschrieben hatte, wechselte er Anfang des Jahres 2000 in die Wirtschaft. Mittels einer Initiativbewerbung fand er gleich bei seinem heutigen Arbeitgeber eine Stelle. In einem firmeninternen Projekt konnte Franz in seinem gewünschten Fachbereich tätig werden und kam da – wie auch privat – in Berührung mit Linux, vorerst allerdings noch als reiner Nutzer.

Nachdem dieses Projekt nach drei Jahren beendet war, machte sich Franz Gedanken über sei-

¹⁴⁵ Der Name wurde geändert.

nen weiteren Werdegang. Aus Interesse und bestehenden Kontakten, aber auch aus karriereorientierten Überlegungen heraus bemühte er sich um einen Wechsel zum Linux-Team.

Da Franz bereits sehr gut mit dem Linux-Kernel vertraut war, konnte er schon nach sehr kurzer Zeit eigenen Code beisteuern, wobei es sich hauptsächlich um Code-Optimierungen und weniger um neue Funktionalität handelte. Aufgrund seiner Fähigkeiten sowie seiner mehrjährigen Firmenzugehörigkeit übernahm er schon bald die fachliche Verantwortung für ein internationales Team mit sieben Mitarbeitenden.

8.3.1.2 Technologische Orientierung

An Open Source schätzt Franz insbesondere die sehr gute Qualität der Software. Maßgeblich ist für ihn weniger das Entwicklungsmodell an sich, sondern vielmehr die klar vorgegebenen Standards. Als Beispiel nennt er die zahlreichen Testfälle, die jedes Mal durchlaufen werden müssen, bevor ein Patch akzeptiert wird. Auch die nicht nur definierten, sondern auch vollzogenen Regeln – beispielsweise für die Codeformatierung – sagen ihm zu. Diese Wertschätzung kommt daher, dass er in seinem vorherigen internen Projekt gegenteilige Erfahrungen gemacht hatte. Sein Vorgesetzter konnte aufgrund der hierarchischen Stellung Code durchsetzen, den Franz als minderwertig beurteilte: „*So war das, so eine Wild-West-Mentalität.*“

Einen weiteren Grund für die aus seiner Sicht bessere Qualität von Open-Source-Software sieht Franz darin, dass sie noch relativ jung ist und (noch) keine Altlasten mitführen muss. Gerade in seiner Firma, die teilweise noch jahrzehntealten Code benutzt, musste Franz Althergebrachtes aufrechterhalten. Insofern kann er es sich vorstellen, dass der Code aus der Open-Source-Community mit der Zeit an Qualität einbüßen könnte, da er auch bei ihr ein zukünftiges Problem von Altlasten vermutet. Zur Zeit gefällt es Franz jedoch, dass er durch die Arbeit am Linux-Kernel sehr nahe am Puls der Zeit ist.

Franz betont, dass er sich in erster Linie aus technischen Überlegungen heraus für die Arbeit am Linux-Kernel entschieden hat. Ob dieses Projekt nun Closed oder Open Source ist, war für ihn eher nebensächlich.

8.3.1.3 Ein Job wie jeder andere

Franz macht seine Arbeit sehr gern, zieht aber klare Grenzen zwischen Beruf und privatem Leben. Er hat kein Bedürfnis, sich zu Hause auch noch an den Computer zu setzen und sieht dasselbe Verhalten auch bei den meisten seiner Kollegen. Obwohl er zumindest teilweise von zu Hause aus arbeiten könnte, gefällt ihm der Kontakt zu den Kollegen am Arbeitsplatz.

Er charakterisiert sich insofern als typischen Informatiker, als er einen guten theoretischen Hintergrund hat. Damit grenzt er sich vom Hacker ab, der sich aus seiner Sicht weniger um Fragen des Designs kümmert, sondern sich eher im Code vertieft. Franz meint, dass beide Typen ihren Platz sowohl in der Open-Source-Community als auch in der Firma haben sollten. Er lässt jedoch mehrfach durchblicken, dass der von ihm geschriebene Code qualitativ hochwertiger ist.

Für die Arbeit am Linux-Kernel sieht Franz keine Probleme. Er musste zwar zu Beginn einen internen Kurs besuchen, in dem ihm erklärt wurde, was er wie machen darf. Auch hat er sich juristisches Basiswissen angeeignet. Der zeitliche Aufwand dafür war jedoch gering. *„Und das ist auch im täglichen Arbeiten kein Hindernis.“* Im Allgemeinen sieht Franz die Probleme, die vor ein paar Jahren bei der Zusammenarbeit der Firma mit der Community noch bestanden haben, weitgehend als behoben an. Dabei haben sich aus seiner Sicht beide Seiten aneinander angepasst.

Die konkreten Aufgaben von Franz stammen aus einem Planungsprozess, der auf der Managementebene ausgearbeitet wird. Zwar kann er selbst Vorschläge einbringen, da diese allerdings ebenfalls denselben Planungsprozess durchlaufen, macht er nur Eingaben, die bereits aus seiner Sicht im Interesse der Firma sind und somit große Chancen haben, akzeptiert zu werden.

Franz erkennt viele Aspekte des Open-Source-Entwicklungsmodells in der Firma. Es gibt Codereviews, Testpläne, Programmierrichtlinien etc. Der entscheidende Unterschied liegt für ihn darin, dass die Community dies viel konsequenter durchsetzt. Er hinterfragt jedoch nicht, weshalb das so ist, sondern akzeptiert es als Tatsache.

Auch die Projektorganisation über das Internet ist für ihn keine Besonderheit von Open-Source-Projekten. Denn aufgrund der Tatsache, dass viele Teams in seiner Firma dezentral,

oft über mehrere Kontinente und Zeitzonen hinweg organisiert sind, ist es für Franz nicht ungewöhnlich, sich asynchron zu organisieren.

Für Franz sind sehr gute technische Fähigkeiten entscheidend für die erfolgreiche Mitarbeit am Linux-Kernel. Diese spricht er sich selber zu. Zwar gäbe es zahlreiche andere Aufgaben, die weniger anspruchsvoll wären, aber *„um da wirklich angesehen zu sein, braucht man ein sehr tiefes, tiefgehendes technisches Verständnis.“* Daneben sei auch viel Geduld und ein dickes Fell für die Arbeit in der Community nötig, da der Umgangston nicht immer angenehm sei. Wer sich jedoch an die Standards und Regeln der Linux-Kernel-Community halte, habe keine ernsthaften Probleme. Franz hat sich denn auch, bevor er zum ersten Mal in der Community aktiv wurde, selbstständig darüber informiert.

8.3.1.4 Wenig Präsenz in der Open-Source-Community

Franz ist eher wenig in der Open-Source-Community aktiv und glaubt nicht, dass er dort bekannt ist. Sein Hauptkontakt zur Community besteht darin, dass er Patches einsendet und dann in die daraus entstehende Diskussion eingebunden ist. Aber *„einfach so mitdiskutieren tue ich relativ selten.“* Eine Maintainer-Funktion hat er nicht und strebt eine solche auch nicht an.

Seine Art der Kommunikation orientiert sich eher an firmeninternen Gepflogenheiten: Als ersten Schritt sucht Franz das persönliche Gespräch. Falls dies nicht möglich ist, nutzt er die firmeneigene Kollaborationssoftware. E-Mails und Mailinglisten nehmen eine eher untergeordnete Rolle ein.

Privat war Franz bisher noch an keinem Open-Source-Projekt aktiv beteiligt. Weil er gerade einen Fehler in einer von ihm verwendeten Software entdeckt hat, macht er sich Gedanken über ein Engagement. Obwohl er seine Erfahrungen mit Open Source in seinem Beruf als durchaus positiv bezeichnet, fällt ihm die Entscheidung nicht leicht. Obwohl das betreffende Projekt nichts mit seiner beruflichen Tätigkeit zu tun hat, würde er zuerst das Einverständnis seiner Firma einholen.

8.3.1.5 Firmeninteressen haben Priorität

Zwar schätzt es Franz, dass der Quelltext von Linux offen ist, hat er doch zumindest privat auch schon negative Erfahrungen mit nicht behobenen Fehlern oder gar aus dem Markt genommener Software gemacht. Aber er stellt geistiges Eigentum und deren Verwertungsrechte nicht infrage. „*Also, diese Ideologie von Richard Stallman, die teile ich nicht.*“ Nach seiner Einschätzung soll seine Firma den vollen Nutzen haben, wenn sie eine Software erstellt.

Auch dass die Firma Termine aus planerischen und marketingtechnischen Gründen vorgeben muss, wird von Franz nicht hinterfragt. Er sieht jedoch die Diskrepanz zur Open-Source-Community, die sich bei ihren Releases nicht am Kalender, sondern an Qualitätskriterien orientiert. Für die Firma und für ihn entsteht dadurch Mehraufwand, den er lieber vermeiden würde. Dass die Firma nur einen sehr beschränkten Einfluss auf die technologische Richtung von Open-Source-Projekten hat, empfindet Franz als störend. Er kritisiert denn auch, dass die Open-Source-Community tendenziell elitär ist und sich weniger an praktischen Überlegungen orientiert.

Unter Freiheit versteht Franz weitgehend die Flexibilität in Bezug auf die Arbeitszeiten, die seine Firma ihm gewährt. Mehr Freiheiten in der Gestaltung des Arbeitsinhalts ist zwar durchaus ein Ziel für ihn. Er akzeptiert jedoch, dass seine Aufgaben durch die Interessen der Firma definiert werden. Jedoch würde er gern in einem frei von ihm gewählten Bereich forschen, und zwar „*unabhängig davon, ob es eine Firma interessiert*“.

8.3.1.6 Den Status quo erhalten

Mit seinem Lohn ist Franz zwar durchaus zufrieden, er kritisiert jedoch die Gehaltspolitik seiner Firma. Interessant ist dabei insbesondere, dass dies die einzige echte Kritik an seinem Arbeitgeber ist. Ansonsten schätzt er die Sicherheit sowie die Möglichkeit, in verschiedene Bereiche Einblick zu erhalten, die ihm eine große Firma bieten kann. Wichtig ist ihm ein möglichst ruhiges Arbeitsumfeld ohne Überraschungen.

Die Spielregeln einer börsennotierten Unternehmung sieht er als systembedingt an und akzeptiert sie weitgehend, ohne sie zu hinterfragen. Zum Beispiel wurde kürzlich seinem Team ein Mitarbeiter zugunsten eines anderen, höher eingeschätzten internen Projektes abgezogen. Dadurch musste er seine eigenen Prioritäten anpassen und eine bereits begonnene Arbeit unter-

brechen. *„Das ist jetzt nicht das erste Mal gewesen. Das wird auch nicht das letzte Mal sein. Und da muss man sich mit arrangieren.“*

Franz ist mit seiner momentanen beruflichen Situation sehr zufrieden. Er hat zwar noch keine konkreten Pläne für die Zukunft, es ist ihm jedoch wichtig, dass er sich möglichst viele Optionen offenhält – und dies, obwohl er gegenüber Veränderungen eher negativ eingestellt ist. Grundsätzlich ist für ihn eine technische Karriere erstrebenswerter als eine in der Führungshierarchie. Die Nähe zur Technologie sähe er aber nicht gefährdet, sollte er sich für einen hierarchischen Karriereschritt entscheiden.

8.3.2 Idealtypische Charakterisierung

„Pragmatische Programmierer erledigen ihre Arbeit, und sie machen es gut.“

(Hunt und Thomas 2003, S. XVIII)

Der Typ des Pragmatischen Ingenieurs traf elfmal zu und ist damit der häufigste in der vorliegenden Stichprobe. Er charakterisiert sich vornehmlich durch eine extrinsische Motivation. Für ihn sind ein guter Lohn, ein sicherer Arbeitsplatz, gute Arbeitsbedingungen und flexible Arbeitszeiten sowie hervorragende berufliche Aussichten dominierende Aspekte bei der Wahl der aktuellen Tätigkeit. Dazu gehört auch, dass Open-Source-Software aufgrund des offenen Quelltexts, des großen Wissenspools sowie dem Peer Reviewing gute Lernmöglichkeiten bietet. Seine extrinsische Motivation zeigt sich darin, dass er sich durch die Veröffentlichung seiner Arbeit zu einer besseren Leistung angeregt fühlt, denn *„das Wissen, dass öffentlich re-viewt wird, reicht schon, dass man sich bemüht, wirklich sauberen Code zu schreiben.“* (B2)

An Open-Source-Software wird vorwiegend die sehr gute Qualität sowie der aktuelle technische Stand geschätzt. Der Pragmatische Ingenieur sieht sich vornehmlich als Ingenieur im engeren Sinne und orientiert sich stark an der Technologie. Über den ideologischen Hintergrund macht er sich kaum Gedanken. Meistens sind die Begriffe Open-Source-Software sowie Freie Software nur oberflächlich bekannt und werden nicht selten gar inhaltlich falsch interpretiert.

Da die meisten der befragten Pragmatischen Ingenieure schon in ihren früheren beruflichen Tätigkeiten in internationalen Teams Software entwickelten, erweist sich das Entwicklungsmodell von Open-Source-Software zumindest in ihrem beruflichen Umfeld kaum als Neuheit,

denn „*insofern ist es vom Arbeitsprofil her [...] nicht so viel anders als eine Projektarbeit an einem anderen Projekt. Wir haben unsere Planung. Wir haben unsere Mitarbeiter. Wir haben unsere Chefs. Wir haben unsere Teamleitung. Das ist alles vorhanden hier.*“ (B2) Sie sprechen einem Open-Source-Softwareentwickler nur wenige besondere Fähigkeiten zu – „*Es ist auch nur programmieren.*“ (B2) – und verorten diese dann insbesondere im kommunikativen Bereich.

Da sie sich vorwiegend für den Linux-Kernel als Software interessieren, empfinden die Pragmatischen Ingenieure das Open-Source-Entwicklungsmodell als eher störend. Insbesondere die mangelnde Kontrolle bei Terminen wird zwar als systembedingt akzeptiert. Sie würden es jedoch begrüßen, wenn die Firma mehr Macht ausüben könnte. Die internen Planungsprozesse werden hingegen nicht infrage gestellt, obwohl die damit verbundenen häufigen Sitzungen als übertrieben empfunden werden und ihrer Ansicht nach oft auch nicht einen direkten Einfluss auf die Qualität ihrer Arbeit haben.

Die Unabhängigkeit der Community wird im eigenen, an Technologie und Qualität orientierten Interesse genutzt, um gegen interne Zwänge zu wirken. „*Ja, für uns Entwickler ist es eigentlich gut, ohne Termine arbeiten zu können.*“ (T20) Damit wird ausgedrückt, dass die Software-Qualität von den ausreichenden zeitlichen Ressourcen abhängig ist. Zusätzliche Zeiteresourcen werden so beim Management nicht selten durch das Argument beansprucht, dass die Community die Qualität noch nicht akzeptiert.

Die rechtlichen Einschränkungen, die durch die Arbeit einer Firma mit der Open-Source-Community entstehen, werden kaum thematisiert. Wichtig ist lediglich, dass die Mitarbeitenden bei eventuellen Problemen durch die Firma abgesichert werden. Störend ist, dass bei Code, der auf Betriebsgeheimnisse zugreift, mit Vorsicht vorzugehen ist, z. B. indem die Rechtsabteilung Abklärungen macht, oder dass „*man [beim Programmieren] manchmal ein bisschen Klimmzüge machen [muss], damit es alles noch im Rahmen ist*“ (F5).

Im Allgemeinen hält sich dieser Typus in der Community so weit wie möglich zurück. Kontakte bestehen weitgehend innerhalb der Firma, das gilt auch für persönliche Gespräche mit Kollegen anstelle des Austausches über Mailinglisten. Dies mag auch daran liegen, dass sich die meisten Interviewteilnehmenden über den oft harten Kommunikationsstil in der Open-Source-Community beklagen. „*Also, manchmal ist der Ton sehr hart oder schroff. Oder*

manchmal auch schon fast verletzend. Das schreckt mich auch ab.“ (E5) Es wird ein rein sachlich argumentierender Stil bevorzugt und auch selbst praktiziert.

Dass der geschriebene Code in aller Regel upstream in den Mainline-Kernel¹⁴⁶ gehen sollte, wird akzeptiert, der dadurch entstehende Mehraufwand würde jedoch lieber vermieden. Der Grund für diese Regel wird darin gesehen, dass der Code nicht mehr allein von der Firma gewartet werden muss und somit Kosten gespart werden können.

Für den Pragmatischen Ingenieur stehen die Bedürfnisse seines Arbeitgebers im Vordergrund. Er sieht die Berechtigung von Linux darin, dass seine Firma einen direkten Nutzen daraus ziehen kann. Der Glaube an den Markt und das freie Unternehmertum wird nie infrage gestellt. Dass direkte Konkurrenten gemeinsam am Linux-Kernel arbeiten, wird eher als problematisch und als potenzieller Interessenkonflikt¹⁴⁷ gesehen. Das Potenzial einer offenen Kollaboration bleibt unbeachtet. Die Bürokratie eines großen, weltweit agierenden Konzerns wird höchstens am Rande thematisiert. Hierarchien werden weitgehend akzeptiert und als Chance für einen eigenen Karriereschritt gesehen. Es ist bezeichnend, dass beinahe alle Pragmatischen Ingenieure, die schon mehrere Jahre in der Firma tätig sind, zumindest Teamleitungsfunktionen innehaben. Der mit der Karriere oft verbundene zunehmende administrative und führungsbezogene Aufwand wird zwar als eher unangenehm wahrgenommen, aber akzeptiert. *„Ich entwickle halt selbst bevorzugt. Ist meine Lieblingstätigkeit. Die anderen beiden sind so das notwendige Übel.“ (D4)* Am liebsten würde sich der Pragmatische Ingenieur den ganzen Tag der Softwareentwicklung widmen. Auch hier zeigt sich die starke Orientierung an der Technik.

Der Pragmatische Ingenieur geht den Konflikten am liebsten aus dem Weg. Dies erreicht er

146 Die Begriffe „Mainline“ und „upstream“ werden weitgehend synonym verwendet. Damit wird ausgedrückt, dass es sich um den offiziell gültigen, aktuellsten Quelltext des Linux-Kernels handelt. Im Gegensatz dazu gibt es unzählige Repositories, sowohl firmenintern als auch frei zugänglich im Internet, die weniger offiziell bzw. aktuell sind.

147 *„Natürlich gibt es Interessenkonflikte. Dass Modul XY von Firma A kommt. Und solche Sachen. Dass Firma B [direkter Konkurrent] von hinten schauen muss, wie sie ihre Sachen irgendwie da rein bekommen. Das ist gerade bei Modul XY vielleicht so eine eigene Sache. Weil, weil der ganze Code, der da im Kernel ist, alles von Firma A geschrieben worden ist. Und Firma B postet da auch direkt auf die Mailinglisten. Es wird dann auch irgendwann genommen. Aber die haben halt immer schlechtere Karten. Was dann teilweise so gemacht wird, ist, über andere Firmen. Ob das jetzt wir sind, oder Firma C oder wer auch immer. [...] Diesen Interessenkonflikt spüre ich vielleicht auch so ein bisschen. Zwischen den Firmen, die eigentlich erbitterte Konkurrenten sind, aber dann doch irgendwie zusammenarbeiten müssen.“ (Interviewauszug I9)*

einerseits, indem er hierarchische Verantwortung nicht nur ablehnt, sondern Probleme nach oben delegiert. Für ihn ist es vorteilhaft, dass die Firma ein Planungs- und Managementgremium hat, das sich vornehmlich um (seine) Konflikte kümmert.

Als größtes Problem in ihrer Arbeit sehen die entsprechenden Befragten, dass sie zu wenig Zeit für ihre Aufgaben haben oder zu wenig Mitarbeitende in ihrem Arbeitsbereich tätig sind. Dies mag darauf zurückzuführen sein, dass sie eher kurzfristige Aufträge erhalten und deshalb oft unter Zeitdruck stehen. Dies stört ihr Ruheempfinden, und sie können nicht nach Plan arbeiten: *„Ich finde es angenehm, wenn es wirklich ruhig ist. Dabei kann ich entspannen und mich auch gut konzentrieren. Und ich organisiere meine Arbeit auch gerne so, dass ich am Stück an einem Projekt arbeite. Möglichst nicht allzu viel durcheinander hab.“ (F6)*

Trotzdem können sich die Pragmatischen Ingenieure gut von der Arbeit abgrenzen und sind der Ansicht, dass sie eine gute Work-Life-Balance aufweisen. *„Denn letztendlich mach ich schon in der Firma sehr viel für den Linux-Kernel. Und da muss ich das nicht auch noch in meiner Freizeit machen.“ (L12)* Mit zunehmendem Alter und zunehmender Erfahrung gleicht sich dieses Verhältnis zusehends aus. Die Familiengründung führt nicht selten zu einer Relativierung der Bedeutung der Arbeit.

Zusammenfassend kann gesagt werden, dass der Pragmatische Ingenieur sehr dem Softwareentwickler entspricht, wie er in proprietärer Softwareentwicklung zu erwarten ist. Er orientiert sich an seiner (technischen) Arbeit und kaum an unternehmerischem Denken. Er sieht sich nicht als etwas Besonderes und grenzt sich mit seinem großen Arbeitsethos sowie seiner angestrebten Effizienz von einem Open-Source-Entwickler ab, der in der Freizeit tätig ist. Markant ist auch, dass alle vier Interviewteilnehmenden, die einen firmeninternen Wechsel von einem Closed-Source-Projekt zur Linux-Kernel-Entwicklung machten, diesem Typus zuzuordnen sind.

8.4 Der dialektische Informatiker

8.4.1 Charakterisierung des Prototyps: Gerhard¹⁴⁸ (G7)

8.4.1.1 Werdegang

Gerhard ist 37 Jahre alt und alleinstehend. Schon seit seiner Kindheit, „*seitdem ich zehn oder elf war*“, beschäftigt er sich mit Computern. Die Wahl des Informatikstudiums lag deshalb nahe. Bereits während des Universitätsstudiums interessierte er sich nicht nur für Open-Source-Software, sondern arbeitete auch in verschiedenen Projekten mit. Nach Abschluss seiner Promotion im Jahr 2000 entschied sich Gerhard gegen eine akademische Karriere und für einen Gang in die Industrie. Dabei war für ihn klar, dass er im Open-Source-Bereich tätig sein wollte.

Gerhard ist seit 2000 beim selben Arbeitgeber angestellt. Im ersten Jahr steuerte er noch Code in den verschiedensten Bereichen des Linux-Kernels bei. Danach fokussierte er recht schnell auf einen Teilbereich, dem er auch heute noch treu ist. Gerhard hatte dafür zu sorgen, den zuerst nur intern vorhandenen Code upstream in den Kernel zu integrieren. Dadurch wurde er offizieller Maintainer dieses Codes, wobei dieser, wie bislang auch, vorwiegend in der eigenen Firma gewartet wird. Jedoch steuerte Gerhard zunehmend Code in seinem Spezialbereich bei. Dies ging über die Bedürfnisse seines Arbeitgebers hinaus. Dieses Engagement hatte zu Folge, dass ihn die Community vor einem Jahr zu einem globalen Maintainer über ein umfassendes Programmpaket gewählt hat. Diese Arbeit als globaler Maintainer sieht Gerhard zwar nicht als einen Auftrag der Firma. Es wird ihm jedoch – weitgehend stillschweigend – erlaubt, einen gewissen Teil seiner Arbeitszeit darauf zu verwenden.

Auch in der Firma nahm der Einfluss zu. Als Architekt und Teamleader kann Gerhard über seinen Bereich zumindest technologisch weitgehend bestimmen, was wie und wann getan wird. Da er seit einiger Zeit auch mit externen Partnern zusammenarbeitet, hat sich dieser Einfluss weit über die Firmengrenzen hinweg ausgeweitet.

Gerhard ist mit seiner beruflichen Entwicklung sehr zufrieden. Er strebt eine technische Lauf-

148 Der Name wurde geändert.

bahn an, die ihm die Firma ermöglicht. Nachdem er bereits ein paar Karriereschritte durchlief, steht zurzeit der nächste, etwas größere an.

8.4.1.2 Etwas bewegen können

Für Gerhard ist es wichtig, dass er mit seiner Arbeit etwas bewegen kann. In seiner heutigen Funktion als Architekt und Teamleader ist dies gegeben, da er für die Architektur und Strategie in seinem Bereich abteilungsübergreifend zuständig ist und technische Entscheidungen weitgehend selbstständig treffen kann. Er nimmt deshalb in Kauf, dass er nun weniger Zeit für das Programmieren hat und oft an Meetings verschiedener Planungsgremien teilnehmen muss. Zudem nahm der Aufwand für die Beantwortung von E-Mails und für administrative Tätigkeiten zu. Um weiterhin aktuelles Wissen zu erlangen, schreibt Gerhard immer wieder selbst Code.

An der Open-Source-Bewegung gefällt ihm die Offenheit des Entwicklungsprozesses. Durch den öffentlich einsehbaren Quelltext und den Review-Prozess wird der Code auch besser lesbar. Somit wird das Ein- und Mitarbeiten einfacher. Zudem können wegen der kaum existierenden Hierarchien alle Beteiligten durch sachliche Argumentation und Engagement aktiv werden und etwas bewegen.

Eine Aufgabe von Gerhard als Teamleader ist es, die in der Arbeit mit Open Source weniger erfahrenen Teammitglieder an die Prozesse und Regeln heranzuführen. Dies ist eine gute Strategie, um potenzielle Konflikte frühzeitig erkennen und diskutieren zu können.

Insbesondere als globaler Maintainer kann er seine Doppelrolle so einsetzen, dass er den von seiner Firma kommenden Code bereits in der Entstehung betreuen kann und so allenfalls später nicht ablehnen muss.

8.4.1.3 Firma und Community zusammenbringen

Gerhard sieht sich in zwei Rollen. Einerseits ist er von der Firma bezahlt und muss ihre Interessen vertreten. Das heißt, dass an Funktionalität gearbeitet wird, die für das eigene Geschäftsmodell von Bedeutung ist. Da der Code jedoch noch upstream gehen soll, gelten auch in der Firmenrolle die Spielregeln der Community. Andererseits muss er als globaler Maintai-

ner die Interessen der Community vertreten. Obwohl er die Arbeit als globaler Maintainer zumindest teilweise während der Arbeitszeit erledigen kann, sind dabei Firmeninteressen tabu. Auffallend ist, dass Gerhard sowohl bei Firmenangelegenheiten als auch in Open-Source-Belangen immer von „wir“ redet, sich also mit beiden Institutionen identifiziert.

Diese beiden Rollen spiegeln sich auch in seiner Einstellung zur Ideologie der Freien Software wider. Grundsätzlich findet er Richard Stallman und dessen *„Standpunkt, der auch gut begründet und fundiert ist“*, begrüßenswert. Und er ist beeindruckt, wie Stallman seine Idee konsequent verfolgt. Allerdings akzeptiert er, dass bei seiner Firma kommerzielle Interessen im Vordergrund stehen und deshalb die Idee nur so weit unterstützt wird, wie es für sie von Nutzen ist.

Gerhard passt seine Kommunikation an die jeweilige Situation an. Firmenintern kommuniziert er oft über die eigene Kollaborationssoftware, was ihm den Kontakt über die verschiedenen Zeitzonen vereinfacht. Ansonsten bevorzugt er E-Mails als Kommunikationsmittel in der Open-Source-Community. Diskussionen, die zwar mehrheitlich intern geführt werden, aber von allgemeinem Interesse sein könnten, versucht er in der Regel, auf öffentlichen Listen zu führen, auch wenn dies mit einem Mehraufwand für ihn verbunden ist.

Für Gerhard entstehen gelegentlich Probleme innerhalb der Firma, wenn er gleichzeitig mit den eher traditionell orientierten Kollegen und den *„Leuten, die so mehr wirklich Hardcore-Open-Source-Entwickler sind“*, zu tun hat. Während erstere zum Beispiel Powerpoint-Präsentationen bevorzugen – wofür auch eine Infrastruktur in der Firma besteht –, wünschen letztere eine einfache, textorientierte Meldung. *„Und da muss man immer so ein bisschen versuchen zu vermitteln. Aber, im Großen und Ganzen klappt das schon.“*

Im Linux-Bereich innerhalb der Firma wurde die Infrastruktur zumindest teilweise an die Bedürfnisse der Open-Source-Entwickler und -Community angepasst. Beispielsweise wird für das Fehlerhandling nicht das in der Firma übliche Bugtracking-System verwendet, sondern das in der Community vorgezogene Bugzilla. Allerdings müssen Mitarbeitende aus anderen Bereichen, die ein Problem mit Linux feststellen, dies in Bugzilla festhalten. *„Und das funktioniert eigentlich ganz gut.“*

Solange sich Gerhard innerhalb des Linux-Bereichs in seiner Firma bewegt, entstehen kaum noch Konflikte zwischen der Open-Source-Arbeit und der kommerziellen Orientierung der

Firma, da sich dieser Bereich durchgängig der Open-Source-Mentalität verschrieben hat. Zudem hat sich die Firma auf höchster Ebene stark zu Linux und Open Source bekannt. *„Aber, manchmal muss man halt darüber hinausgehen.“* So bereitete zum Beispiel die Tatsache, dass bei Projekten der Free Software Foundation eine Abtretung der Rechte am beigetragenen Code unterschrieben werden muss, der Rechtsabteilung der Firma Kopfzerbrechen. Die Klärung dieses Problems dauerte zwar ein gutes Jahr und forderte auch von Gerhard viel Engagement und Geduld. Aber wenn für die Firma ein gangbarer Weg gefunden ist, gilt dieser als Standard ohne weitere Diskussionen.

Wie beim Pragmatischen Ingenieur entstehen auch hier immer wieder Probleme, wenn die Firma gegenüber Partnern und Kunden Termine vereinbart, die durch die Prozesse in der Open-Source-Community nicht eingehalten werden können. Gerhard sieht dies als Herausforderung, die es zu handhaben gilt. Manchen Problemen kann er vorbeugen, etwa indem er, wenn ein Patch nicht sofort akzeptiert wird, im Budget eine Reserve für „community-bedingten“ Mehraufwand einplant.

Dies zeugt davon, dass er versucht, Konflikte zu vermeiden, indem er rechtzeitig mit anderen zusammen eine allgemein akzeptable Lösung sucht. Die juristischen Restriktionen, die aus der Arbeit einer großen Firma mit der Open-Source-Community entstehen, sieht Gerhard zwar als größten Störfaktor, als Behinderung empfindet er sie jedoch nicht. *„Konflikte gibt es immer. Aber ich denke, es läuft im Großen und Ganzen eigentlich ganz gut.“* Für die in einer großen Firma systembedingt entstehende Bürokratie, die im täglichen Arbeiten eher störend ist, sieht er durchaus gute Gründe und akzeptiert sie.

8.4.2 Idealtypische Charakterisierung

Der Typ des Dialektischen Informatikers sieht sich nicht mehr in der Rolle des Programmierers, sondern versteht sich als Vermittler zwischen dem globalen Konzern und der Open-Source-Community. Dabei versucht er stets, vorhandene Gegensätze aufzuheben. Der dialektische Informatiker hat sich bereits während seiner Studienzzeit intensiv mit Open-Source-Software beschäftigt und nicht selten seine Abschlussarbeit zu diesem Thema geschrieben. Er hat auch darauf hingearbeitet und eine entsprechende Anstellung gesucht, mit der er im Open-Source-Bereich tätig sein kann. Dass dabei gerade der Linux-Kernel im Zentrum steht, war

meist nicht zwingend und könnte sich in Zukunft durchaus wieder ändern.

Er ist vorwiegend extrinsisch motiviert, wobei für ihn die Reputation bei Gleich- und Höhergestellten sehr wichtig ist. Diese Anerkennung versucht er vorwiegend mittels exzellenter Leistung zu erlangen, hierarchische Anerkennung ist ihm weniger wichtig. Die Open-Source-Welt bietet ihm aufgrund der Transparenz hier besondere Möglichkeiten. *„Jeder sieht, was ich gemacht habe. Wenn ich es gut mache, sieht man es. Wenn ich es schlecht mache, sieht man es auch.“* (C3) Dass in die Reputation auch investiert werden muss, ist ihm klar. Er kennt seine Grenzen und ist bereit, Hilfe von anderen zu akzeptieren. Es erscheint ihm für das Arbeiten in der Open-Source-Community wichtig, dass man mit Kritik umgehen kann, sowohl mit derjenigen, die man erhält, als auch mit derjenigen, die man selbst formuliert. Insofern ist *„nicht jeder, der gut programmieren kann [...] auch ein guter Open-Source-Entwickler.“* (U21)

Der Dialektische Informatiker ist durch seine Integrität und seine Bereitschaft, für seine Meinung einzustehen, sowohl in der Community als auch innerhalb der Firma anerkannt. Für das Wirken in der Community will er ein klares Bekenntnis der Firma zur Open-Source-Software. Dadurch dass er von einer großen Firma beschäftigt wird, sieht er keine direkten Vorteile in der Community. Er schätzt jedoch, dass er so mehr Möglichkeiten hat, mit anderen *„großen Firmen zu kommunizieren“* (J10). Umgekehrt sieht er seine Position in der Firma durch seine Reputation in der Community, nicht selten als Maintainer, gestärkt. Er sieht sich sowohl in der Firma als auch in der Community sehr gut integriert.

Er verwendet sehr viel Zeit auf Kommunikation, sei es firmenintern in Sitzungen oder in der Community mittels E-Mails. In der sachlichen Kommunikation, insbesondere auch beim Peer Reviewing, sieht der Dialektische Informatiker die Möglichkeit, etwas bewegen zu können, was ihm sehr wichtig ist, sowohl innerhalb der Firma als auch in der Community. Aus diesem Grunde akzeptiert er auch, dass nur noch wenig Zeit zum Programmieren bleibt. Dafür hat er sich jedoch Freiraum geschaffen, um *„am Ball zu bleiben“* (G7). Seine Aufgaben sind dabei eher längerfristiger, strategischer Natur. *„Wir als Architekten und technische Experten haben natürlich einen entscheidenden Einfluss darauf, was inhaltlich denn zu tun ist.“* (G7)

Vor diesem Hintergrund kann er sich gut den Werten und Normen sowohl der Firma als auch der Community anschließen. Er passt sich der jeweiligen Rolle an und wechselt seinen Stand-

punkt und sein Verhalten entsprechend. Wo er Konflikte sieht, versucht er zu vermitteln und einen für beide Seiten akzeptablen Weg zu finden. Typisch für den Dialektischen Informatiker ist es insbesondere, dass er möglichen Konflikten vorbeugt, indem er selbstständig den Kontakt zu den Betroffenen sucht.

Er sieht in der Firma eine Entwicklung hin zum Open-Source-Stil. Die Open-Source-Community hat sich aus seiner Sicht durch Selbstorganisation bereits sehr gut an die Zusammenarbeit mit den Firmen angepasst.

Dass der geschriebene Code in aller Regel upstream in den Mainline-Kernel gehen sollte, begrüßt er. Den dadurch entstehenden Mehraufwand sieht er als gute Investition an. *„Wenn wir jetzt auf dem Weg, um das Ziel zu erreichen, noch andere Sachen machen müssen, ist das ja kein Problem für uns. Sondern, das können wir gerne mitmachen. Das ist dann halt was, wo ich dann halt die Zeit dann auch mit einplanen muss, für dieses Projekt.“* (H8) Den Grund für diese Regel sieht er weniger in Kostenersparnissen, sondern in der strategischen Bedeutung, dass der Linux-Kernel einerseits dadurch kontinuierlich besser wird und damit gegenüber der proprietären Konkurrenz bestehen kann. Andererseits sieht er ein, dass langfristig die Wartung von eigenem Code aufgrund der häufigen Veröffentlichungen und zahlreichen Distributionen schwierig wird. Diese Haltung vertritt er auch gegenüber seinen Auftraggebern, seien sie aus der Firma, seien es externe Partner oder Kunden. *„Also, generell sagen wir unseren Partnern, wenn sie etwas [...] haben möchten, müssen sie es upstream in der Community akzeptiert haben. Also sprich, wir probieren, die Partner davon zu überzeugen, dass sie aktiv mit der Community zusammenarbeiten.“* (H8)

Die Arbeit einer kommerziell orientierten Firma mit einer im Grunde nicht profitorientierten Community sieht der Dialektische Informatiker nicht als störend, sondern vielmehr als Herausforderung. Von seinem Arbeitgeber wird von ihm *„erwartet, einen Ausweg zu finden“* (H8), wenn z. B. ein gewünschtes und für das Geschäft als wichtig eingestuftes Feature von der Community nicht akzeptiert wird. Wie er die Lösung findet, ist eine *„Frage von Balance“* (J10) zwischen Firmen- und Community-Ansprüchen. Die mangelnde Kontrolle seiner Firma vor allem in Bezug auf die Termine versucht er zu handhaben, indem er mögliche Probleme frühzeitig in der Planung berücksichtigt und so selbst weniger unter Druck gerät. Am meisten stören ihn Beschränkungen, die ihm von juristischer Seite auferlegt werden. Ein Beispiel dafür ist das kollaborative Arbeiten:

„Zum Beispiel, eine der Kernideen von Open Source ist ja die gemeinschaftliche Arbeit. Dass jemand mal einen Patch schreibt. Und schickt den an eine zweite Person, die daran dann etwas ändert. Und dann wird es weitergeschickt. Und so weiter. Und dass der Code also durch viele Hände geht. So. Und jetzt hat die Rechtsabteilung hier ein gewisses Problem damit, wenn ein Angestellter Code von jemand anders bekommt, dann was arbeitet, und den dann in der Gesamtheit wieder nach Außen schickt. Weil, es könnte ja sein, dass in dem Code, den wir von jemand anders bekommen haben, zum Beispiel ein Patent von irgendjemand vielleicht verletzt hat. Das weiß man ja nie. Und wir schicken ihn im Namen der Firma weiter raus. Dann kommt natürlich von uns Code, der vielleicht ein Patent verletzt. Es entsteht also ein mögliches Risiko für die Firma. [...] Deswegen ist das immer ein gewisses Konfliktpotential. Weil eben die sozusagen aus der Open-Source-Sicht naheliegendste Methode, wie wir arbeiten wollen und sollen, eben aus Sicht der Firma unnötige Risiken verursacht.“ (G7)

Unterstützt wird der dialektische Informatiker in seiner Arbeit in der eigenen, auf Open-Source-Software spezialisierten Organisationseinheit und auch vom höheren Management. Beim mittleren Management, das eher weniger mit der Community zu tun hat, versucht er Verständnis für die besonderen Belange seiner Arbeit zu generieren, unterstützt sie aber auch dort, wo dies nicht gelingt.

Seine Kontakte findet der Dialektische Informatiker sowohl innerhalb der Firma als auch in der Community. Dabei ist für ihn die Firmenzugehörigkeit unbedeutend. Das führt dazu, dass berufliche Kontakte in der Freizeit weitergeführt werden. Umgekehrt nutzt er sein Netzwerk in der Community, um sich allenfalls mit Partnern oder gar Konkurrenten informell über technische Fragen auszutauschen. *„Weil ich da halt auch privat Freunde habe bei Partnern, und halt auch bei Konkurrenten. Also das ist da sehr vernetzt, quasi. Man tauscht sich da ja nicht über die Produktsachen aus, aber halt wenn es um Technologien geht, entsteht da schon Austausch. [...] Also normalerweise hat man da schon, ja, so ein Netzwerk aufgebaut, von Leuten die man halt kennt, die man halt fragen kann. Das ist eigentlich auf der Ebene von der direkten Kommunikation einfacher, meistens, an Informationen zu kommen.“ (H8)*

Die Unabhängigkeit, Offenheit und Freiheit der Software ist für ihn wichtig, wobei er dies pragmatisch sieht und seiner Firma das Recht zugesteht, proprietäre Software zu entwickeln,

wenn sich dies anbietet. Der Dialektische Informatiker versteht die beiden Welten nicht als Gegensatz oder Konkurrenz, sondern sieht die Berechtigung beider Typen und glaubt, dass sie gut nebeneinanderher gehen oder sich im Einzelfall sogar gegenseitig befruchten können. Er akzeptiert nicht nur, dass die meisten Entwickler in der Community von Firmen dafür bezahlt werden, sondern sieht das eher als Vorteil, da so mehr und vielfältiger Code ins Projekt einfließt. Nicht selten arbeitet er zusätzlich auch in seiner Freizeit an Open-Source-Software, wobei dies eher weniger mit seinen konkreten beruflichen Aufgaben zu tun hat. *„Ein anderer Effekt, der viel häufiger ist, sind Leute, die von Firmen bezahlt werden, aber in ihrer Freizeit an Sachen mitarbeiten, für die sie nicht bezahlt werden. Das mach ich genau so. [...] Da ist dann jemand bei einer anderen Firma zuständig für ein eigenes Projekt, und der schickt mir einen Patch zu meinem Projekt. Und ich kenne den schon von anderen Sachen, wo wir zusammen arbeiten, aber das ist dann für ihn sein Hobby erstmal.“ (C3)*

Der dialektische Informatiker kann die Grenze zwischen Beruf und Privatem gut ziehen. Er arbeitet zwar sehr viel, aber ohne dass der Zeitaufwand außer Kontrolle gerät. Hilfreich dabei ist, dass er Spaß an der Arbeit hat und sie ihn befriedigt. Allerdings erkennt er auch, dass es Wichtigeres als seine Arbeit gibt, zum Beispiel die Gründung einer eigenen Familie. *„Das hat sich am 28. Januar 2007 quasi von selbst verändert. Weil, da wurde mein Sohn geboren. [...] Wenn ich nach Hause komme – ich nehme es mir zwar immer vor, und heute Abend machst du jetzt noch das und das –, aber wenn ich dann zu Hause bin, dann gibt es auf einmal Wichtigeres.“ (H8)* Das bedeutet, dass der berufliche Status durch private Ereignisse relativiert wird. Veränderung wird demnach in das Leben integriert.

8.5 Der sozialromantische Hacker

8.5.1 Charakterisierung des Prototyps: Robert¹⁴⁹ (R18)

8.5.1.1 Werdegang

Robert ist 36 Jahre alt, Single und Ende der 1990er Jahre aus dem angrenzenden Ausland für

¹⁴⁹ Der Name wurde geändert.

die heutige Anstellung zugezogen. Er schloss eine Berufslehre als Betriebselektriker ab und begann bereits während der Lehrzeit und in der Freizeit zu programmieren. Über sein Hobby ist er später zum professionellen Softwareentwickler geworden.

Schon sehr früh, zu Beginn der 1990er Jahre stieß Robert auf Linux. Er war damals bereits professioneller Programmierer. Da in seiner damaligen Firma mit UNIX gearbeitet wurde und er auch privat ein zumindest UNIX-artiges Betriebssystem verwenden wollte, kam für ihn Linux gerade zur richtigen Zeit. Die Vorteile von Linux gegenüber den meisten anderen UNIX-Derivaten waren für ihn, dass die Software auf einem normalen PC lief und kostenlos war.

Bald installierte er die Software auch in der Firma und konnte durch die bessere Performance und die einfachere Wartung Kollegen und Management von Linux überzeugen. In seinem nächsten Job konnte Robert von seinem weitgehend aus Büchern im Selbststudium erworbenen Wissen sowie durch seine privaten Experimente mit Linux profitieren. Er baute sein Wissen zu Linux kontinuierlich aus.

Während einer Phase der Arbeitslosigkeit, ungefähr um 1995, begann Robert sich noch intensiver mit Linux zu beschäftigen. Er kompilierte den Kernel zum ersten Mal selbst und spielte mit einer minimalen Kernel-Konfiguration herum. Tage- und nächtelang arbeitete er an einer Lösung für seine Probleme und sendete schließlich sein Resultat an die Community. Das positive Feedback darauf ermutigte ihn weiterzumachen. Aufgrund dieser Arbeiten wurde Robert von seinem heutigen Arbeitgeber für eine Anstellung angefragt. Überzeugt von Linux wagte er den Schritt und wechselte den Wohnsitz ins Nachbarland.

8.5.1.2 Eine neue emotionale Heimat wird gesucht und gefunden

Robert hatte vor seinem Wechsel nach Deutschland mehrere eher kürzere Anstellungen und dazwischen erwerbslose Phasen. Der Grund für die Aufgabe einer Anstellung lag dabei weniger in der eigentlichen Tätigkeit, sondern im ideellen und menschlichen Umfeld. Zuletzt, als er vermehrt mit Windows und der dazumal noch proprietären Programmiersprache Java beschäftigt war, wuchs die Abneigung gegen Microsoft, worauf er im Linux-Bereich eine Tätigkeit suchte. Überzeugt an seinem neuen Arbeitgeber hat ihn letztlich, dass es „menschlich“ passte. *„Ich fand das halt einfach menschlich auch sympathisch, das war genau die Richtung,*

die mir gefällt.“

Die Zugehörigkeit zu einer Gemeinschaft ist für Robert zentral, auch bei der Arbeit. Umso wichtiger ist dies, da er durch den Wegzug aus seiner Heimat sein gewohntes soziales Umfeld verlor. Seine Arbeitskollegen dienten ihm als Familienersatz und bildeten am neuen Ort weitgehend seinen Freundeskreis. Den Bereich seiner Firma – ein internationaler Konzern – sieht er vorerst als abgeschlossene, selbstständige Einheit, dem er sich emotional stark verbunden fühlt. Robert findet seine Community somit im Firmenumfeld und weniger in der Open-Source-Gemeinschaft.

Die Open-Source-Community versteht Robert eher als abstrakt. Er verbindet damit weniger Personen, sondern vielmehr die Idee der „Geschenkultur“ und entspricht damit dem marxistisch orientierten Hacker, wie ihn Raymond vor allem in seinem Essay „Homesteading the Noosphere“ charakterisiert (Raymond 1999). Dass sein Arbeitgeber auch Geld verdienen muss, blendet er dabei weitgehend aus.

8.5.1.3 Zunehmende Entfremdung in der Arbeit

Als Robert 1999 seine heutige Anstellung antrat, war Linux ökonomisch noch von geringer Bedeutung für die Firma. Vieles war noch im Entstehen. Strukturen und Regeln gab es kaum, die Hierarchie war flach und Meetings fanden selten statt. Vieles basierte auf Eigenverantwortung und Eigeninitiative. *„Es war wirklich wie eine Open-Source-Community, in der Firma auch.“*

Der Linux-Bereich war dabei weitgehend abgeschottet von den Strukturen eines internationalen großen Konzerns. Mit der zunehmenden Bedeutung von Linux innerhalb der Firma wurde der Bereich jedoch immer stärker in die firmenweit standardisierten Strukturen und Prozesse integriert. Die von Robert gesuchte und gefundene Sozialromantik ging dabei zusehends verloren.

Auch mit den Spielregeln eines börsennotierten Unternehmens bekundete Robert, Probleme zu haben. Er konnte seine kapitalismusfeindliche Haltung nicht verbergen und störte sich insbesondere an den aus seiner Sicht oft ungerechtfertigten Entlassungswellen. *„Und das tut irgendwie, das ist, das hat mir sehr weh getan, jedes Mal.“* Dabei störte ihn nicht nur, dass seine Vorstellung von Gerechtigkeit und Zweckmäßigkeit verletzt wurde, auch das ständige

Auseinanderbrechen seiner „Familie“ machte ihm zusehends zu schaffen.

Im Gegensatz zu vielen seiner Kollegen hat sich für ihn dabei auch das Betriebsklima erheblich verschlechtert, was ihn resignieren lässt. *„Aber, ja man bekommt halt sozusagen sein Geld. Und das ist halt dann das am Ende, was noch übrig bleibt.“* Aufgrund der zunehmenden Entfremdung von der Firma hat sich Robert immer wieder einen Stellenwechsel überlegt. Trotzdem ist er der Firma bisher neun Jahre treu geblieben. Dies schreibt er den sozialen Kontakten zu, obwohl sich diese kontinuierlich verflüchtigen. Entscheidend für das Verbleiben ist, dass Robert über die Jahre hinweg eine starke Verbundenheit zum Arbeitsumfeld aufgebaut hat und sich dadurch nur schwer abgrenzen kann. Eine Kündigung würde für ihn somit einen tief greifenden Bruch in der persönlichen Geschichte bedeuten.

8.5.1.4 Ausweichen auf andere Schauplätze

Bereits früh versuchte Robert, den zusehends schwindenden Idealismus für seine Arbeit zu kompensieren. Als Kontrast zur zunehmenden Standardisierung in seinem Bereich sucht er eine an die Arbeit angelehnte, aber nicht direkt damit verbundene Aktivität. Dies führte allerdings zur Doppelbelastung, was sich durch eingeschränkte Leistungsfähigkeit am Arbeitsplatz und letztlich durch einen Burn-out¹⁵⁰ deutlich bemerkbar machte. Unterstützt von seinen Kollegen und mithilfe einer durch die Firma organisierte Psychologin entwickelte er eine Strategie, wie er seinen Burn-out meistern kann. *„Und, ja, die Entscheidung war dann, ein Jahr unbezahlten Urlaub zu nehmen.“* Von der Firma wurde für das Jahr eine Stelle ausgeschrieben, und Robert hatte das schriftliche Versprechen bekommen, in derselben Abteilung wieder eingestellt zu werden.

Zurück an der Arbeit hatte er zwar sein Tief überwunden, die Diskrepanz zwischen erwünschtem und tatsächlichem Arbeitsumfeld blieb jedoch bestehen. Dies führte zu einem Ablösungsprozess. *„Einfluss hat die Firma nicht mehr sozusagen, den Stand einer Familie für mich, wie sie es früher gehabt hat. Also insofern hat sich sozusagen die Beziehung gelockert.“* Dieser Ablösungsprozess fand jedoch nur sehr langsam statt. Einerseits machte ihm die Arbeit an sich immer noch sehr viel Spaß, und das Arbeitsumfeld bot doch mehr Freiheiten, als er bei einem anderen Arbeitgeber erwartet. Zudem war die gewohnte Umgebung eine gewisse Si-

150 Eine kurze Einführung zum Thema „Burn-out“ in der IT-Branche bietet z. B. Kreft (2008).

cherheit, die er nach den erlebten Tiefschlägen nicht ersatzlos aufgeben wollte. Erst eine Liebesbeziehung, die zu Ende gegangen war, eröffnete eine neue Orientierungsphase. „*Nachdem ich halt hier sehr viel mich mit Firmendingen beschäftigt habe, [...] also hatte mein Leben halt nicht stattgefunden, [...] in den letzten paar Jahren.*“ Robert besann sich wieder vermehrt seiner Wurzeln. Er erkannte, dass ihm der Arbeitsort kulturell nicht entspricht. Eine Rückkehr in sein Heimatland wird für ihn somit zunehmend erstrebenswerter. Auch die Gründung einer eigenen Familie zieht er – nun, da er seine soziale Familie in der Firma verloren hat – ernsthaft in Betracht.

Er hatte bereits mit seinem Vorgesetzten die Möglichkeit besprochen, dass er in sein Heimatland zurückkehren und dort Telearbeit machen möchte. Zum Zeitpunkt des Interviews war er jedoch noch nicht bereit, diesen Schritt tatsächlich zu vollziehen.

Robert schloss nicht aus, sich vollständig aus der professionellen Softwareentwicklung zurückzuziehen. Bezeichnend für ihn ist, dass er ein zweites Mal den Traum ins Auge fasst, sein Hobby zum Beruf zu machen: In den letzten Jahren hatte er mit Fotografieren begonnen und könnte sich gut vorstellen, trotz des erwarteten Einkommensverlustes ein Fotostudio mit Fotogeschäft zu eröffnen. Nicht ganz zufällig ist wohl die Tatsache, dass er damit sein eigener Herr und Meister und somit nicht mehr abhängig von den Entscheidungen von Managern wäre, die emotional und räumlich weit von ihm entfernt sind.

8.5.2 Idealtypische Charakterisierung

Der Typ des Sozialromantischen Hackers lässt sich charakterisieren durch eine weitgehend intrinsische Motivation. Dabei ist die Zugehörigkeit zu einer Gemeinschaft zentral, an deren Werten und Normen er sich orientiert. Prägnant beim sozialromantischen Hacker ist, dass er sich zuerst eine Gemeinschaft aussucht, die seinen moralischen Wertvorstellungen entspricht, und sich erst dann ihren Normen entsprechend verhält.

Der Sozialromantische Hacker sieht sich in der Gemeinschaft weitgehend selbst verwirklicht. Er hat dadurch allerdings Probleme, sich als eigenständige Person abzugrenzen. Dies kann bis zur Selbstaufopferung und dem damit verbundenen Burn-out führen. Dieser Altruismus ist nicht am Wohl anderer Individuen, sondern am Gemeinwohl orientiert und somit idealistisch. Er orientiert sich somit an der marxistisch geprägten „Geschenkultur“ (Raymond 1999).

Beim Sozialromantischen Hacker kann somit eine kapitalismusfeindliche Haltung festgestellt werden, welche sich einerseits im Kampf gegen proprietäre Software – „*ich hasse das Windows*“ (R18) – manifestiert. Andererseits zeigt er kein Verständnis für firmenpolitische Entscheidungen und mag hierarchische Strukturen nicht.

Dass die Arbeit Spaß machen muss, kann in der Softwareentwicklung vorausgesetzt werden und ist in der vorliegenden Studie bei allen Teilnehmenden gegeben. Beim Sozialromantischen Hacker ist der Spaßfaktor zwar ebenfalls gegeben, allerdings fällt auf, dass er im Vergleich zu den anderen Typen weniger ausgeprägt ist.

Typisch am Sozialromantischen Hacker ist, dass er sich im Umfeld eines großen Konzerns in einem Dilemma zwischen seinen eigenen Wertvorstellungen und den gewachsenen Normen einer börsennotierten Firma befindet. Während es anderen Programmierern mit ähnlichen Wertvorstellungen gelingt, sich mit diesen Normen zu arrangieren, hängt dieser Typ seinen idealen Vorstellungen der unabhängigen Softwareentwicklung weiterhin nach. Um sein Dilemma zu lösen, muss er aus diesen Normen ausbrechen. Dies kann er nur erreichen, indem er den Job oder gar den Beruf wechselt, was erklären mag, wieso im vorliegenden Setting nur eine einzige Person dem Typus entspricht. Wie im oben beschriebenen Prototyp führt ein Ausweichen auf ein anderes Betätigungsfeld neben der Arbeitstätigkeit nicht zum Erfolg, da sich der Sozialromantische Hacker kaum von seiner Arbeit abgrenzen kann.

Der Typ des Sozialromantischen Hackers ist nur ein einziges Mal aufgetreten. Auffallend ist, dass es sich um die einzige Person ohne einen Tertiärabschluss handelt. Das vorhandene Wissen basiert ausschließlich auf der Praxis sowie auf dem Selbststudium. Ob dies Zufall ist oder ob ein Zusammenhang besteht, lässt sich aufgrund der kleinen Stichprobe nicht abschließend beurteilen.¹⁵¹

151 Das entsprechende Interview war insgesamt speziell. Als einziges wurde es auf Wunsch des Interviewpartners außerhalb der Büroräumlichkeiten und bei einem gemeinsamen Nachtessen geführt. Obwohl er den Termin vergessen hatte, war er aber um 19.30 Uhr noch am Arbeitsplatz telefonisch erreichbar. Ob das ausführliche und offene Gespräch auf die Umgebung oder den Charakter des Interviewpartners zurückzuführen ist, kann nicht abschließend beantwortet werden. .

Teil IV: Diskussion, Schlussfolgerungen & Ausblick

9 Diskussion

Dass Linux und Open-Source-Software im unternehmerischen Einsatz von immer größer werdender Bedeutung sind, ist in verschiedenen Newsforen nahezu täglich nachzulesen und wird auch von zahlreichen Studien und namhaften Beratern belegt.¹⁵² Bedeutend weniger Informationen sind über die Produktionsseite der Software erhältlich. Dass die Produktion im Falle des Linux-Kernels mittlerweile mehrheitlich durch Firmen vorangetrieben wird, konnte in dieser Studie anhand einer quantitativen Untersuchung der Linux-Kernel-Beiträge aufgezeigt werden. Durch leitfadengestützte Interviews mit Linux-Kernel-Entwicklern in großen IKT-Firmen wurde zudem die kommerzielle Perspektive der Open-Source-Software-Entwicklung vertieft. Die qualitative Analyse brachte anhand der Dimensionen Motivation sowie Normen- und Wertesystem drei unterschiedliche Typen von Entwicklern hervor, die in den Unternehmen am Linux-Kernel mitarbeiten: den Pragmatischen Ingenieur, den Dialektischen Informatiker sowie den Sozialromantischen Hacker. Ein aufgrund der Dimensionen möglicher vierter Typ, der Hedonistische Programmierer, war in der Stichprobe hingegen nicht anzutreffen.

Im Folgenden wird in einem ersten Teil diskutiert, dass große IKT-Firmen die treibende Kraft hinter Open Source sind, und es wird gefragt, weshalb andere Studien weniger deutliche quantitative Ergebnisse aufweisen. Sie bevorzugen darüber hinaus eher eine politisch bzw. ideell motivierte Argumentation, obwohl sich die Open-Source-Bewegung mit ökonomischen Modellen erklären lässt und sich in aktuelle ökonomische Trends einfügt. Die Analyse des Linux-Kernels lässt zudem einige strukturelle Folgerungen zu. Es bietet sich in diesem ersten Diskussionsteil abschließend an, den Infrastrukturbegriff aufgrund der gewonnenen Erkenntnisse zu erweitern.

Im zweiten Teil wird das für Open-Source-Projekte typische Basar-Modell um die Komponente der Firmenbeteiligung erweitert. Die Motivation der Beteiligten muss unter dem

152 Insbesondere die Finanzkrise hat ab Herbst 2008 der Open-Source-Bewegung neuen Schwung gegeben: So bietet die Suche nach „open source' finanzkrise“ auf Google insgesamt 110.000 Treffer [Suche am 02.07.2009]. Die Analysten von Gartner hatten jedoch bereits im Februar 2008 Open Source als eine der Top-Ten-Entwicklungen der kommenden Jahre prognostiziert. Sie haben dabei vorausgesagt, dass Unternehmen, die diesen Trend verschlafen, einen entscheidenden Wettbewerbsnachteil haben werden. Nur zwei Monate später erklärten die Gartner-Analysten in der Studie „The State of Open Source 2008“, dass sie erwarten, dass bis 2012 mehr als 90 % der Unternehmen Open Source direkt oder indirekt (eingebettet in Hard- oder Software) verwenden werden (die Studie wurde in zahlreichen Newsforen zitiert, z. B. unter <http://www.zdnetasia.com/news/software/0,39044164,62039870,00.htm> [17.08.2009]).

Aspekt, dass ein Großteil von ihnen von Firmen für ihre Beiträge bezahlt werden, neu diskutiert werden. Die aufgrund der Interviewergebnisse erstellte Typologie wird in das Rollenkonzept der Communities integriert, und die daraus erwachsenen Wechselwirkungen werden diskutiert. Erläutert wird auch, wie die Typen unterschiedlich mit Problemen umgehen und welche Veränderungen bei den Typen feststellbar sind. Abschließend wird die Selbstorganisation der Community um die Komponente der Firmenbeteiligung erweitert.

9.1 Firmenbeteiligung bei FLOSS

9.1.1 Ökonomische Interessen

Die quantitativen empirischen Daten des Linux-Kernels zeigen auf, dass ein Großteil der Beiträge von kommerziell orientierten Firmen stammt: Dass 73 % aller Beiträge kommerziell motiviert sind, ist ein deutliches Resultat.¹⁵³ Auch in den Interviews wurde diese Tatsache bestätigt, sind doch alle Interviewteilnehmenden der Meinung, dass sie fast ausschließlich mit Entwicklern anderer Firmen in der Community zusammenarbeiten. Ähnliche Ergebnisse liefert eine Auswertung des Eclipse-Projektes (Spaeth et al. 2008), bei der neben den Codebeiträgen auch die Beiträge in den Foren und Mailinglisten analysiert wurden.

Beide Studien widersprechen jedoch der in Anwender- und Fachkreisen weit verbreiteten Annahme, dass FLOSS weitgehend von Freiwilligen produziert wird. Sie stehen zudem im Widerspruch zu anderen Forschungsergebnissen, insbesondere der MERIT-Studie (Ghosh 2006), der „*weltweit sorgfältigsten Studie*“ (Lutterbeck et al. 2007, S. 5) zur ökonomischen Relevanz von FLOSS. Die MERIT-Studie geht von einem Firmenanteil an den gesamten Codebeiträgen von lediglich knapp 20 % aus, die individuellen Beiträgen beziffert sie hingegen mit über 60 % (Ghosh 2006, S. 50).

Wie lässt sich diese Diskrepanz erklären? Die meisten Forschungsprojekte, die wie die MERIT-Studie auf einem Projektmix bestehen, beziehen ihre Daten aus einem zentralen Re-

¹⁵³ Entsprechende Einschätzungen zum Linux-Kernel hat u. a. Alan Cox (Red Hat) in seiner Key Note auf der OpenExpo im März 2008 getroffen, <http://video.google.de/videoplay?docid=1893415028065590416> [17.08.2009], oder Robert S. Sutor (IBM, Vice President Open Source and Standards) in seiner Key Note auf der Veranstaltung Open Source Meets Business im Januar 2008 in Nürnberg.

pository, in der Regel dem größten Open-Source-Repository SourceForge. Dies ist aus Sicht der Datenerhebung zwar sehr praktisch, da sich damit Hunderttausende von Projekten mit weitgehend homogenen Daten analysieren lassen. Allerdings werden auf dieser Basis die großen Projekte, die eine eigene Plattform unterhalten, vernachlässigt. Untersuchungen von solchen Projekten mit einer entsprechenden Verbreitung haben gezeigt, dass der Anteil von Firmenbeteiligungen sehr hoch ist. Zudem befinden sich viele der Projekte auf Plattformen wie SourceForge in einem „Dornröschenschlaf“; sie werden nicht mehr oder nur sehr langsam weiterentwickelt und ihre Community besteht – wenn überhaupt – nur aus sehr wenigen Personen. Daraus schließt Karl Fogel pointiert: „*Most free software projects fail.*“ (Fogel 2006, S. 1)¹⁵⁴

Die MERIT-Studie basiert auf der Linux-Distribution Debian, die eine ausgewogene Mischung von großen und kleinen, professionellen und Freizeit-Projekten enthält und somit als repräsentativ bezeichnet werden kann. Auch hier gilt: Von den über 8.500 in Debian enthaltenen und in der Analyse berücksichtigten Softwarepaketen ist nur ein sehr kleiner Prozentsatz aus volks- wie auch aus betriebswirtschaftlicher Sicht relevant.¹⁵⁵

Hinzu kommt die Problematik der Zuteilung der Codebeiträge zu den Firmen. Zum einen wurden bei der Analyse des Debian-Projekts gemäß Beschreibung ausschließlich die Source-Dateien ausgewertet. Darin stehen jedoch in der Regel nur der Ersteller bzw. der Maintainer des jeweiligen Files sowie in gewissen Projekten die einzelnen Autoren, aber nicht, wer im Detail welche Zeilen beigetragen hat. Oft wird diese Funktion des Erstellens bzw. Maintainens von Freiwilligen wahrgenommen oder von Angestellten, die jedoch diese Funktion unabhängig von ihrer Anstellung ausführen. Dieses Vorgehen verfälscht das Resultat. Um genauere Angaben zu generieren, müssen die einzelnen Bewegungen im jeweiligen Repository analysiert werden, d. h. die Commits¹⁵⁶. Dies ist jedoch ein Vorgehen, das aufgrund der Da-

154 Wie eine empirische Studie von Healy und Schussman (2003) anhand des Open-Source-Repositorys SourceForge aufzeigt, hatten von den gut 45.000 untersuchten Projekten 25 % noch gar keine Software geschrieben und 95 % hatten maximal 5 registrierte Beitragende. Als Konsequenz werden nur gut 2 % der Projekte als gereift eingestuft. Positiv betrachtet kann SourceForge aus dieser Perspektive als riesiger Ideenpool gesehen werden, von denen sich die besten dann durchsetzen werden.

155 Entsprechende Details zu Debian sind unter der URL <http://libresoft.es/debian-counting/sarge/> [10.08.2009] abrufbar. So haben z. B. über 3.000 Pakete noch nicht den Zustand einer stabilen Version 1 erreicht und mehr als 5.000 Pakete beinhalten weniger als 10.000 Codezeilen (was in etwa Linux in der Version 0.01 im Jahre 1991 nach nur einem halben Jahr Entwicklung durch Linus Torvalds entspricht).

156 In einer Versionsverwaltung versteht man unter einem „Commit“ das Hochladen einer Codeänderung in das zentrale Repository.

tenmenge kaum zu bewältigen ist. Es erscheint dem Verfasser deshalb sinnvoller, einige repräsentative und gezielt ausgewählte Projekte anhand noch zu bestimmender Kriterien zu analysieren und basierend darauf die Resultate zu aggregieren. Dieses Vorgehen wählte auch O'Mahony (2003) mit sechs ausgewählten und weit verbreiteten Projekten (GNU, Linux Kernel, Apache Webserver, Debian Linux Distribution, die Desktop-Oberfläche GNOME sowie die Linux Standard Base) und erkannte bereits in einer frühen Phase einen Anteil an bezahlter Beteiligung von rund 66 %. Aktuell existieren keine Studien mit diesem Ansatz, sei es durch die eigene Datenerhebung oder eine Metastudie. Die hier vorliegenden Daten des Linux-Kernels können deshalb eine Lücke im derzeitigen Wissensstand schließen und einen Beitrag für eine mögliche Metastudie leisten.

Eine zusätzliche Schwierigkeit bei der Analyse einer solch großen Anzahl von zudem noch heterogenen Projekten besteht darin, die Spezialitäten zu berücksichtigen. Die im Rahmen dieser Studie durchgeführte Analyse des Linux-Kernels hat jedoch ergeben, dass rund ein Drittel der Firmenbeiträge nur durch eine zusätzliche manuelle Bearbeitung sowie durch zusätzliche Datenerhebung mittels individueller E-Mail-Anfrage eruiert werden konnten. Eine Vernachlässigung dieses Aspektes führt folglich zu einer Verschiebung des Resultats hin zu einer überdimensionierten Relevanz der freiwillig Beitragenden.

Zusammenfassend kann festgehalten werden, dass die ökonomische Relevanz von FLOSS über die großen und im unternehmerischen Umfeld verbreiteten Projekte und nicht über einen möglichst breiten Mix eruiert werden sollte, da Erstere trotz ihrer relativ geringen Anzahl aufgrund ihrer Verbreitung von größerer ökonomischer Bedeutung sind. Und bei diesen Projekten zeigt sich eindeutig – was anhand der Analyse des Linux-Kernels im Kapitel 7.2 belegt werden konnte –, dass Freiwilligkeit im Sinne von unbezahlter Arbeitsleistung eine untergeordnete Rolle spielt.

9.1.2 Ökonomische vs. ideelle Argumentation

Die Aussage der MERIT-Studie (Ghosh 2006), dass 70–80 % der Beiträge von Freiwilligen im Sinne eines bürgerschaftlichen Engagements geleistet werden, scheint nicht nur metho-

disch bedingt, sondern auch stark politisch motiviert zu sein. Da die Studie auf der Verwertungs-Seite belegt, dass Open-Source-Software einen enormen ökonomischen Gewinn generieren kann, versuchen die Studienverfasser so, die Unterstützung freier und offener Software für die starke wirtschaftliche Lobby in der EU attraktiv zu machen. Die Aussicht, mithilfe einer Armada an freiwillig und unentgeltlich Mitarbeitenden einen Profit zu erzielen, greift jedoch zu kurz. Einerseits haben die Interviewergebnisse aufgezeigt, dass der Kontrollverlust durch die Mitarbeit in einer sich selbst organisierenden Gemeinschaft sowohl für die Firmen als auch für die Firmenmitarbeiter ein Problem darstellt, wie das exemplarisch der Prototyp des Pragmatischen Ingenieurs zeigt. Dieser Kontrollverlust kann systembedingt¹⁵⁷ nur durch kontinuierliche und substanzielle Beiträge in einem akzeptablen Rahmen gehalten werden. Dahlander und Wallin (2006) beschreiben dies mit dem Konzept „a man on the inside“. Anhand einer empirischen Studie des GNOME-Projekts untersuchen sie mittels einer quantitativen Analyse der Kommunikation auf den Mailinglisten über mehrere Jahre hinweg die Bemühungen von Firmen, ihre Mitarbeiter ins Zentrum des Netzwerkes zu stellen, damit diese das Projekt mit beeinflussen können. Die vorliegende Studie kann dieses Konzept aufgrund der quantitativen Analyse des Linux-Kernels sowie der qualitativen Interviews bestätigen.

In der MERIT-Studie wird zudem ein Unterschied von quelloffener und proprietärer Software ausgeblendet: Durch Lizenz, Copyright und oft auch durch Patente können Softwarefirmen eine monopolartige Position oder zumindest ein Oligopol kreieren, in dem die Margen höher sind. Dies ist bei Open-Source-Software aufgrund der freien Verfügbarkeit des Codes nicht möglich. Daraus wäre zu schließen, dass Firmen proprietären oder zumindest in seinen Freiheiten stark eingeschränkten offenen Code bevorzugen. Wie die Beispiele von u. a. IBM und SUN zeigen, kann diese These nicht mehr vorbehaltlos unterstützt werden. Dies führt zur Frage, womit diese Firmen (und mit ihnen viele andere) ihre Gewinne erzielen.

Eng verbunden mit dieser Fragestellung ist das Kostenargument. Nutzung und Entwicklung von FLOSS reduzieren die Kosten der individuell Beteiligten, so eine allgemein verbreitete These. Deshalb ist der meistgenannte Grund für den Einsatz von FLOSS in Firmen die Kosteneinsparung (Forrester Consulting 2008; Madsen 2009). Auf der Nutzerseite lässt sich dies

¹⁵⁷ Open-Source-Communities funktionieren in der Regel meritokratisch, das bedeutet, dass Einfluss nur durch Leistung und nicht durch eine hierarchische Machtposition gewonnen werden kann (siehe dazu auch Kapitel 3.2.1).

aufgrund der entfallenden Lizenzkosten relativ leicht begründen und auch belegen.¹⁵⁸ Allerdings divergieren aufgrund des Migrationsaufwandes die TCO („Total Cost of Ownership“) nicht ganz so deutlich, deshalb rechnet sich der Wechsel zu FLOSS meist erst nach mehreren Jahren.¹⁵⁹ Dass es dennoch genügend Firmen gibt, die sich in nicht geringem Ausmaß an der Entwicklung von FLOSS beteiligen, zeigt sich in der Analyse der Linux-Kernel-Logfiles deutlich.

Wie die Interview-Ergebnisse aufzeigen, liegt ein ebenso wichtiger Grund für die aktive Beteiligung darin, dass die jeweiligen Firmen für sie wichtigen Code dem Linux-Kernel beifügen wollen – sei dies für die Unterstützung von Hard- bzw. Software oder zum Erstellen von für das eigene Geschäftsmodell wichtiger Funktionalität. Die Beschreibung der konkreten Tätigkeit der Interviewten aller Typen lässt vermuten, dass die Anzahl der aus einem konkreten, praktischen Bedarf beigetragenen Codezeilen weitaus größer ist als diejenige aus einer strategischen oder ideellen Motivation. Die Strategie zur Unterstützung von FLOSS folgt demnach viel eher der Realität der Bedarfserfüllung, als dass die Beteiligung aufgrund einer definierten Strategie aufgebaut wird.

Hinzu kommt, dass ein Betriebssystem wie Linux hauptsächlich aus Funktionalität besteht, die sich aus Nutzersicht zu konkurrierenden Produkten nicht differenziert, da sie vorausgesetzt werden muss. Ein Betriebssystem, das z. B. nicht von der Festplatte lesen und darauf schreiben kann, dürfte in den meisten Anwendungsfällen nur von geringem Nutzen sein. In diesem Teil muss folglich ein erheblicher Aufwand durch einen Anbieter betrieben werden, ohne dass er sich am Markt abgrenzen kann (Perens 2007). Der Wert eines Betriebssystems

158 Konkrete Angaben über Kosteneinsparungen durch die Migration zu Open-Source-Software macht das Auswärtige Amt in Deutschland (Auener 2008), das insbesondere auch im Vergleich zu anderen Ämtern sehr gut abschneidet. In der Schweiz beziffert der Kanton Solothurn die Einsparungen in der IT durch eine Migration zu einem Open-Source-Desktop mit mehr als 100.000 € jährlich. Dabei darf jedoch nicht verschwiegen werden, dass die IT-Kosten nur die eine Seite sind, aufgrund diverser Kommentare in Benutzerforen darf vermutet werden, dass zumindest ein Teil der IT-Einsparungen auf Anwenderseite durch nicht optimale Software wieder wettgemacht wird. Die Stadt Wien berechnete, dass der jährliche Unterhalt einer Open-Source-Lösung klar günstiger kommt als eine reine Microsoft- oder hybride Lösung. Dagegen sieht es bei den Migrationskosten gerade umgekehrt aus, was die Verantwortlichen zum Entschluss einer sanften Migration geführt hat (siehe dazu die Projektstudie, publiziert unter <http://www.wien.gv.at/ma14/pdf/oss-studie-deutsch-langfassung.pdf> [17.08.2009]). Zwar sind Kosteneinsparungen sehr wichtig bei der Entscheidungsfindung zur Migration zu Linux, der Vorteil der Unabhängigkeit wird in vielen Fällen jedoch stärker gewichtet, wie insbesondere das Beispiel der Stadt München belegt (Details sind nachzulesen unter http://www.osor.eu/case_studies/declaration-of-independence-the-limux-project-in-munich [17.08.2009]).

159 Konkrete Zahlen dazu bietet zum Beispiel eine Studie der Fraunhofer-Gesellschaft (Renner et al. 2005), die ein Einsparungspotenzial beim TCO über sechs Jahre von ca. 2,4 % ermitteln.

liegt zum einen in der Stabilität, Sicherheit und generell in der Qualität. Hier hat sich gezeigt, dass das Open-Source-Prinzip der vielen Augen dem proprietären Modell klar überlegen ist. Zum anderen liegt der Wert in der breiten Unterstützung von Hard- und Software. Hier entsteht durch die Nachfrage im Markt ein Druck auf die entsprechenden Anbieter wie z. B. Intel, IBM oder Oracle. Die Unterstützung von Linux wird von den Kunden gefordert. Eine breitere Unterstützung führt wiederum zur vermehrten Anwendung und dadurch wieder zu mehr Druck aus dem Markt auf die Anbieter – ein Prozess, in dem sich Linux befindet. Deshalb wird das Projekt stetig vorangetrieben. Dieser Prozess entspricht der von Kuwabara (2000) eingeführten evolutionären Entscheidungstheorie (siehe Kapitel 3.2.2.4), bedarf jedoch einiger Anpassungen, um den Anforderungen der kommerziell Agierenden zu entsprechen (siehe Abb. 30).

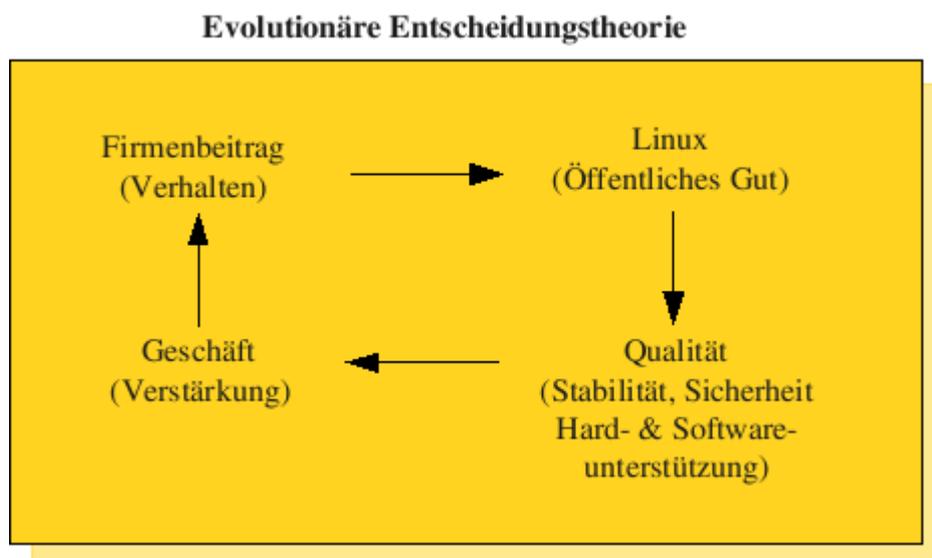


Abbildung 30: Vergleich rationale und evolutionäre Entscheidungstheorie aus Firmensicht in Anlehnung an Kuwabara (2000)

Während das Prinzip des Feedback-Loops dasselbe ist, unterscheiden sich die einzelnen Schritte des Zyklus inhaltlich. Tragen die individuell Beitragenden durch ihr Verhalten vor allem die Werte und Normen der Community, so sind die bezahlten Entwickler in viel größerem Ausmaß verantwortlich für die Qualität der Software. Bei den Individuen findet durch die Reputation eine Verstärkung im Engagement statt, bei den Firmen führt ein zunehmendes Geschäft mit Linux auch zu einer zunehmenden Unterstützung.

Entscheidend ist, dass sich Firmen wie IBM oder Red Hat wegen ihres geringen Beitrags an Code („nur“ 10 %) auf diejenigen Bereiche konzentrieren können, die für sie einen wirklichen Mehrwert und einen Wettbewerbsvorteil bringen. Diese Strategie wird offensichtlich auch nicht durch die Offenlegung des Codes verhindert, was oft als Nachteil und Hemmnis von Open-Source-Software aufgeführt wird. Zwar standen die Firmeninteressen nicht im Zentrum dieser Studie. Aufgrund der Interviewergebnisse gibt es jedoch keinerlei Anzeichen, dass der von direkten Konkurrenten beigetragene Code im Linux-Kernel systematisch untersucht wird. Verschiedentlich wurde erwähnt, dass es mithin Probleme bereitet, Code zu schreiben, der auf betriebsinterne Informationen zugreifen muss.

Dass die einzelnen Architekturen, wie in Kapitel 7.5 aufgezeigt, nicht ausschließlich durch die jeweiligen Hersteller unterstützt werden, zeigt eindrücklich, dass in diesem durch kommerzielle Interessen geprägten Bereich die Community – mit wenigen Ausnahmen¹⁶⁰ – funktioniert, wenn auch fast unter Ausschluss von nicht kommerziellen Institutionen und Privatpersonen. Dass es dabei einzelne Unternehmen gibt, die sich in ihrem Beitrag zum Linux-Kernel teilweise sogar zu hundert Prozent auf ihre eigene Architektur beschränken, ist nicht negativ, da Linux insgesamt durch die möglichst breite Hardwareunterstützung nur gewinnen kann. Andererseits deutet das eher diversifizierte Vorgehen der untersuchten Firmen darauf hin, dass diese die Open-Source-Software und insbesondere das Betriebssystem Linux vermehrt strategisch betrachten.

Es steht folglich nicht eine Senkung der Entwicklungskosten im Zentrum des Firmen-Engagements beim Linux-Kernel¹⁶¹, vielmehr soll der Aufwand dort betrieben werden, wo sich die Firma einen strategischen Vorteil am Markt verspricht. Durch die Vielzahl der beteiligten Firmen entsteht so in einem Mosaik der unterschiedlichen Interessen ein komplettes, marktgerechtes Produkt. Eine Argumentation „Pro-FLOSS“ kann folglich nicht nur auf der Nachfrage-, sondern auch auf der Anbieter- und Produzentenseite ökonomisch geführt werden, ohne dass auf ideelle, moralische oder politische Motivationen zurückgegriffen werden muss.

160 s390/IBM, Sparc/Sun, Blackfin/Analog Devices, Xtensa/Tensilica, CRIS/Axis Communications, AVR32/Atmel und M32R/Mitsubishi

161 Gemäß Ghosh (2006, S. 50) beziffert IBM seine Aufwendungen für die Linux-Entwicklung auf 100 Millionen US-Dollar jährlich.

9.1.3 FLOSS und ökonomische Trends

Jeremy Rifkin (2000) schreibt in seinem Buch „Access: Das Verschwinden des Eigentums“, dass viele Firmen ihr Geschäftsmodell zunehmend nicht mehr auf der Produktion von Waren aufbauen, sondern auf einem zusätzlichen Mehrwert aus dem Produkt für den Käufer. Mit dieser Aussage geht der Abbau des Eigentums an den Produktionsmitteln einher. Erläutert wird dies u. a. am Beispiel der Firma Nike, die ihre Artikel von anderen Herstellern produzieren lassen und ihren Mehrwert aus der Kreation des Designs und eines Lifestyles beziehen. Einerseits kann mit diesem System eine höhere Marge erzielt werden, andererseits wird dadurch eine größere Flexibilität für die Geschäftstätigkeit erreicht. Der Nachteil ist jedoch ein erheblicher Kontrollverlust durch das unvollständige Produktionssystem¹⁶², welcher bisher hauptsächlich durch die Verlagerung der Produktion in weniger entwickelte Länder und das daraus resultierende Machtgefälle weitgehend aufgefangen werden kann. Dass dies ein moralisch bedenkliches Vorgehen sein kann, deckt Naomi Klein in ihrem Buch „No Logo!“ (2001) deutlich auf.¹⁶³

Rifkin bezieht sich nicht auf die Informationstechnologie, sondern auf die herstellende Industrie. In der IKT-Branche steht das Outsourcing für eine analoge Entwicklung des Auslagerns der Produktion in Billiglohnländer, wobei in der Regel die Eigentumsrechte nicht verlagert werden. Jedoch lässt sich mit der FLOSS-Bewegung und der dabei erkennbaren kräftigen Unterstützung durch die IKT-Branche derselbe Prozess des Abbaus des Eigentums verfolgen. Allerdings besteht bei FLOSS der Unterschied darin, dass die Produktion nicht in Billiglohnländer verschoben, sondern in der Allmende durchgeführt wird. Aufgrund der Lizenzbedingungen von Freier Software führt dies unmittelbar dazu, dass auch die so hergestellte Software der Allgemeinheit gehört und nicht mehr durch die Beitragenden privatisiert werden kann.¹⁶⁴ Die daraus resultierenden erheblichen strukturellen und

¹⁶² Ein Produktionssystem, das mehr Kontrolle gewährt und trotzdem nicht auf einer vollständigen Produktionskette beruht, wurde von Toyota ab den 1950er Jahren entwickelt und kontinuierlich perfektioniert (Ohno 1993). Dieses System weist durchaus Parallelen zur Open-Source-Bewegung auf (Evans und Wolf 2005).

¹⁶³ Eine gegenteilige Ansicht vertritt u. a. Bhagwati (2008). Er argumentiert, dass die Globalisierung aufgrund oftmals diffuser Ängste für vieles verantwortlich gemacht wird, was sie gar nicht zu verschulden hat. Er hebt auf der positiven Seite insbesondere den steigenden Wohlstand und die höhere Schulbildung in vielen Ländern der sich entwickelnden Welt hervor und sieht in der Globalisierung nicht die Ursache, sondern einen Beitrag zur Lösung weltweiter Probleme.

¹⁶⁴ Im Gegensatz zur hergestellten Software kann jedoch das Know-how privatisiert werden. Auf dieser Tatsache bauen zahlreiche Firmen ihr Geschäftsmodell mit Open-Source-Software auf.

organisatorischen Veränderungen sind zwar nicht speziell für das Open-Source-Entwicklungsmodell, sie können in ihrem Ausmaß und ihrer Konsequenz jedoch als wegweisend bezeichnet werden:

„Eine allgemeine Tendenz zu Dezentralisierung, Abbau von Hierarchien und offenen Systemgrenzen ist auf jeden Fall nicht zu übersehen. [...] Die Projekte der freien Software übertreffen jede management- und organisationstheoretische Vision an Dezentralisation, lockerer Kooperation und zwangloser Koordination.“
(Grassmuck 2002, S. 332)

Diese Tendenz von FLOSS, die auch als Prototyp von Open Innovation bezeichnet werden kann (West und Gallagher 2006), findet sich in der ökonomischen Literatur wieder. Die Wissensgesellschaft, wie sie von Bell (1976), Drucker (1993) sowie in jüngerer Zeit von Castells (2001) beschrieben wurde, findet ihre Entsprechung im Open-Source-Entwicklungsmodell. Malone (2004) erläutert zudem, dass der Grad der Dezentralisation stark mit der Entwicklung der Kommunikationstechnologie zusammenhängt. Er unterscheidet dabei vier Kategorien von Organisationen auf einem Kontinuum zwischen zentralisiert und dezentralisiert.

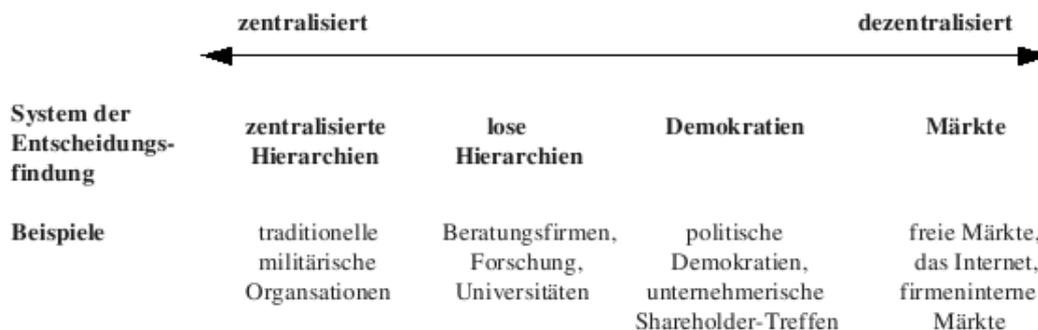


Abbildung 31: Dezentralisierungsgrad (nach Malone 2004, S. 6)

Die von Malone entsprechend Abbildung 31 am einen Ende der Achse Zentralisation–Dezentralisation angesiedelten Märkte entsprechen dabei dem von Raymond (1999) eingeführten Basar-Entwicklungsmodell. Die Achse ist gleichzeitig als Zeitlinie zu lesen. Nicht nur die Open-Source-Bewegung – wie im zweiten Teil dieser Dissertation bereits beschrieben – ist ohne das Aufkommen des Internets nicht denkbar. Auch die rasante Globalisierung der Wirtschaft wurde in der heutigen Breite erst durch die Möglichkeiten des Internets realisierbar.

Daher ist sie auf dem rechten Pol des Kontinuums anzusiedeln. Insofern kann Open Source auch als eine Art der Globalisierung gesehen werden, die – kombiniert mit dem Abbau des Besitzanspruchs durch kommerziell orientierte Organisationen – zu einer Produktion in der Allmende geführt hat.

Dass es sich bei der Open-Source-Entwicklung in weiten Teilen um eine mit aktuellen betriebs- und volkswirtschaftlichen Theorien erklärbare Bewegung handelt, haben die Interviewergebnisse aufgezeigt. Insbesondere der Pragmatische Ingenieur, aber auch der Dialektische Informatiker sehen ihre Tätigkeit durchaus als konform mit der Closed-Source-Entwicklung. Dass es Unterschiede gibt, wird keineswegs negiert – darauf wird im folgenden Kapitel 9.2 näher eingegangen –, aber die Interviewteilnehmenden nehmen diese weitaus weniger wahr als in der bisherigen wissenschaftlichen Diskussion aufgezeigt.

Beim Open-Source-Entwicklungsmodell handelt es sich nicht um einen Sonderfall – und schon gar nicht um eine marxistisch orientierte Kapitalismusfeindlichkeit –, sondern um eine logische Folge allgemeiner Trends in Ökonomie und Technologie. Demnach kann bei der FLOSS-Bewegung nicht von einer Revolution gesprochen werden, sondern vielmehr von einer Evolution.

9.1.4 Strukturelle Folgerungen

9.1.4.1 Digitaler Graben

Die Behauptung, durch Open-Source-Software könne der digitale Graben verringert werden, lässt sich zumindest in Bezug auf die Beteiligung von Firmen durch die Analyse des Linux-Kernels nicht bestätigen. Dominierend sind in den USA domizilierte Firmen. In Europa nimmt Deutschland eine führende Rolle ein. Während Südamerika – trotz des zum Teil klaren öffentlichen Bekenntnisses zu Open Source, wie z. B. durch die brasilianische Regierung (Richter 2006) – und Afrika praktisch nicht vertreten sind, ist der asiatische Raum insbesondere mit Taiwan und Japan rege beteiligt. Einschränkend ist festzuhalten, dass es sich bei den generierten Daten um den Standort der (Mutter-)Firma handelt und nicht um den Standort des Entwicklers. Es kann also durchaus sein, dass mehr Lohnzahlungen in Entwicklungs- und

Schwellenländer fließen, als die Analyse vermuten lässt. Gewinne aus dem Geschäft mit Linux dürften jedoch kaum in größerem Ausmaß in diese Regionen fließen.

FLOSS hat jedoch Vorteile gegenüber proprietärer Software, um den Abbau des digitalen Grabens zu fördern, etwa den Wissenstransfer, geringere Beschaffungskosten, technologische Unabhängigkeit sowie die Lokalisierung in Bezug auf Sprache und regionale Einheiten (Richter 2006). Dass diese Vorteile jedoch nur in geringerem Masse tatsächlich genutzt werden, zeigt die MERIT-Studie (Ghosh 2006) anhand des Beispiels Brasiliens auf, einem oft genannten FLOSS-„Musterland“. Während die Verbreitung von Linux auf Servern über 50 % beträgt, kann sich andere Open-Source-Software kaum durchsetzen. Zurückgeführt wird dies hauptsächlich auf mangelhafte Benutzerschnittstellen.¹⁶⁵ Eine vermehrte Beteiligung von Firmen aus Ländern der sich entwickelnden Welt könnte jedoch zu einem beschleunigten Ausgleich führen. Erste Anstrengungen in diese Richtung gibt es, etwa die virtuelle Zusammenarbeit von 16 afrikanischen Universitäten, die seit 2009 einen FLOSS-Masterstudiengang anbieten.¹⁶⁶

9.1.4.2 Unternehmerischer Mittelstand

Im Gegensatz zu Firmen aus der sich entwickelnden Welt beteiligen sich kleine und mittelständische Firmen überproportional am Linux-Kernel. Zwar kommt der Großteil der Autoren und Codezeilen von relativ wenigen größeren, multinationalen Firmen, jedoch bilden kleine und mittelständische Firmen – vor allem in Europa und insbesondere Deutschland¹⁶⁷ – die Mehrheit der beteiligten Firmen. Dieses Resultat deckt sich weitgehend mit der MERIT-Studie (Ghosh 2006) und wird z. B. durch das deutsche Auswärtige Amt bestätigt (Auener 2008). Die Aussage, dass Open-Source-Software das lokale Gewerbe fördert, kann demnach durch die Analyse des Linux-Kernels bestätigt werden, wenn auch die Mehrzahl der Linux-Arbeitsplätze (rund zwei Drittel) in großen und sehr großen, meist multinational agierenden Konzernen zu finden ist. Hinzu kommt die Tatsache, dass die meisten großen Firmen mehrere

165 Zu beachten gilt es jedoch, dass diese Aussagen auf Daten aus den Jahren 2002 und 2003 basieren und deshalb nicht mehr ganz der Realität entsprechen dürften. Diese Praxis, dass aufgrund von mangelnden aktuelleren Daten eine veraltete Basis verwendet wird, ist gerade im Zusammenhang mit FLOSS oft anzutreffen.

166 Siehe dazu den Artikel unter http://www.epo.de/index.php?option=com_content&view=article&id=5150:it-in-afrika-pioniere-der-freiheit&catid=99:topnews [17.08.2009].

167 Die aktive Rolle von Europa und insbesondere Deutschland bei Linux hat bereits eine lange Tradition, wie z. B. die empirische Studie von Dempsey et al. (1999) aufzeigt.

Entwickler stellen, was auf die strategische Bedeutung von Linux und Open Source schließen lässt.

Die Software- und Dienstleistungsbranche trägt im Jahr 2007 über die Hälfte der Codezeilen bei, mehr noch als die Hardwarevendors. Zusammen lieferten sie über 95 % des Codes, die restlichen 5 % werden durch die Telekommunikationsbranche, den Handel und IT-ferne Branchen geleistet. Dies zeigt deutlich, dass Linux eine Domäne der Hard- und Softwarevendors ist, obwohl es auch anderen Branchen offenstehen würde.¹⁶⁸ Dies könnte sich in Zukunft ändern, wenn auch Non-IT-Firmen sich einen strategischen Vorteil durch den aktiven Einsatz von Open-Source-Software versprechen, wie es aktuell zum Beispiel BMW tut oder wie es sich mit dem angekündigten „Light Car – Open Source“¹⁶⁹ bzw. dem bereits realisierten „Urban Car“¹⁷⁰ abzuzeichnen beginnt.

Dass nicht nur die sehr großen, etablierten IT-Firmen etwas bewegen können, zeigt insbesondere das Beispiel von Red Hat als größtem einzelnen Beitragenden. Der Bereich des Netzwerks sowie des Filesystems wird gar mit jeweils über 20 % der total beigetragenen Codezeilen durch diese eine Firma stark vorangetrieben. Interessant ist ebenfalls, dass die kommerziell orientierten Firmen sich nicht nur um die Hardwareunterstützung kümmern, sondern auch beim eigentlichen Kern sowie beim „Diversen“ wie etwa der Dokumentation 70–80 % der Codezeilen beitragen. Kleine und sehr kleine Firmen fokussieren stark auf die Entwicklung von Treibern. Gut zwei Drittel ihrer Codebeiträge können diesem Bereich zugerechnet werden.

9.1.5 Erweiterung des Infrastrukturbegriffs

Wie in Kapitel 3.4.2 beschrieben, wird unter Infrastruktur Software verstanden, die für das Funktionieren einer Organisation notwendig ist. Dabei kann unterschieden werden zwischen systemnaher Software (insbesondere das Betriebssystem) und Basis-Anwendungssoftware. Der untersuchte Linux-Kernel ist der systemnahen Software zuzurechnen. Zahlreiche Studien,

168 Dass die kommerziellen Firmen bei der Architektur dominierend sind, liegt neben den kommerziellen Interessen auch daran, dass das Know-how der Systeme hauptsächlich, wenn nicht gar ausschließlich, bei den Herstellerfirmen liegt.

169 Siehe dazu z. B. http://www.autobild.de/artikel/edag-light-car-in-genf-2009_833566.html [17.08.2009].

170 Siehe dazu z. B. <http://www.golem.de/0906/67826.html> [17.08.2009].

die auf Daten von Projekten basieren, die diesem Infrastrukturbegriff entsprechen, stützen die in dieser Studie präsentierten Ergebnisse (Roberts et al. 2006; Dahlander und Wallin 2006; Spaeth et al. 2008). Eine Generalisierung der Resultate lässt sich somit im Rahmen des Infrastrukturbereichs vertreten.

Die Frage stellt sich nun, ob sich eine Generalisierung auf FLOSS ausweiten lässt, die nicht diesem Infrastrukturbegriff entspricht? Die Ansicht des Verfassers ist grundsätzlich „Ja“, sofern man den Infrastrukturbegriff erweitert. Unter der Berücksichtigung der Argumentation von Perens (2007), der Software nicht aufgrund von Applikationstypen klassifiziert, sondern eine Einteilung eher auf dem Nutzen der Software aufbaut, kann auch der Infrastrukturbegriff erweitert werden. Software soll gemäß Perens – unabhängig davon, ob sie frei oder proprietär ist – nur zu einem sehr kleinen Teil einen unmittelbaren Einfluss auf den Erfolg eines Betriebes haben. Der Wert der Software liegt demnach primär in der Anwendung des Nutzers. Perens schätzt, dass nur rund 10 % der in einem Unternehmen eingesetzten Software sie von ihren Konkurrenten differenziert und somit einen Teil zur Wertschöpfung beiträgt. Die restlichen 90 % sind demnach nicht differenzierend und, so wird an dieser Stelle vorgeschlagen, können als Infrastruktur angesehen werden:

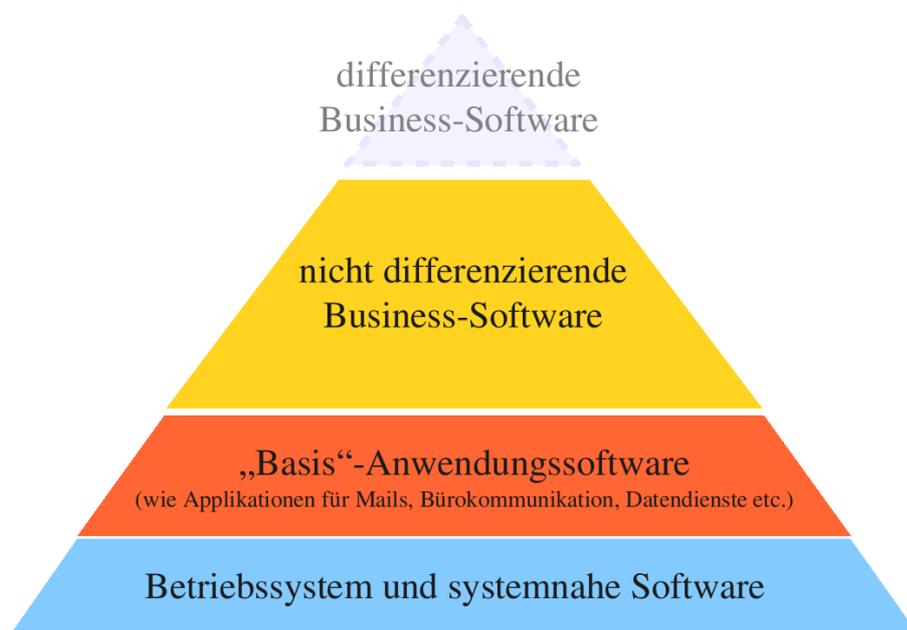


Abbildung 32: Erweiterte Software-Infrastruktur

Die Abbildung 32 legt nahe, dass das Potenzial für eine markante Marktdurchdringung durch

FLOSS weitaus größer ist als gemeinhin und in Anbetracht der Arbeiten bei Linux, Apache, Firefox und anderen angenommen. Wie das Beratungsunternehmen Forrester, so wagt auch Gartner in seiner aktuellen Studie „Open Source Software Impact on IT Services Purchasing Patterns, 2008“ die Prognose, dass bis 2012 annähernd ein Drittel der gesamten IT-Ausgaben in FLOSS-Aktivitäten fließen wird. Die Aussage gilt auch für Europa und Deutschland, wie eine aktuelle Studie aufzeigt:

„Freie Unternehmensanwendungen sind laut beiden Marktforschern seltener im Einsatz, zeigen jedoch einen positiven Trend: Open Source erobert zunehmend auch höhere Ebenen im Softwarestack der Unternehmen.“ (Diedrich 2009)

Auch das im Dezember 2008 in Paris erstmals durchgeführte OpenWorldForum sieht in seiner Roadmap bis 2020 (Laisné 2008) die Adoption von Software durch FLOSS in allen Bereichen als gegeben (siehe Abb. 33).

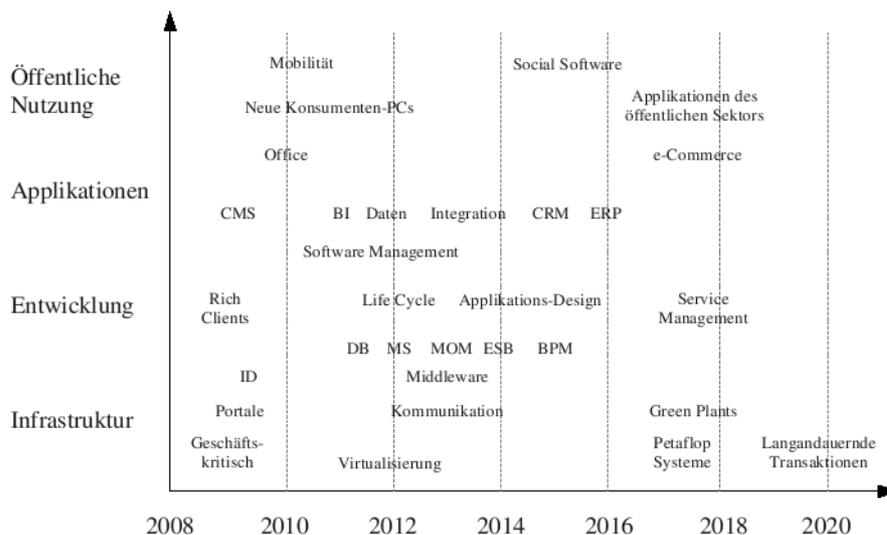


Abbildung 33: Adoption durch FLOSS (nach Laisné 2008, S. 9)

Diese Aussagen sind nicht allzu überraschend. Denn insgesamt besteht Einigkeit darüber, „dass FLOSS die überlegene Methode der Softwareentwicklung benutzt, um diese Infrastruktur bereit zu halten“ (Lutterbeck 2007, S. 3; siehe dazu auch Bauer und Pizka 2005).

Definiert man „Infrastruktur“ nicht aus technischer, sondern aus Nutzen-Sicht, so ist das Potenzial für FLOSS im unternehmerischen Einsatz weitaus größer als bisher prognostiziert. Die in dieser Studie präsentierten Ergebnisse des Linux-Kernels lassen sich demnach mit gewissen Einschränkungen auf eine Vielzahl von Projekten übertragen.

9.2 Ein erweitertes Basar-Modell

Bei der Diskussion um den Anteil von kommerziell orientierten Firmen darf nicht übersehen werden, dass auch diese Beiträge aus Sicht der Open-Source-Projekte freiwillig geleistet werden. Das bedeutet, dass Firmen wie Individuen denjenigen Code beisteuern, der für sie von Bedeutung ist. Demnach verfolgen die Open-Source-Projekte weniger einen Masterplan, sondern sie entwickeln sich vielmehr im Sinne eines Mosaiks mit einzelnen Teilchen kontinuierlich weiter. Was in den Worten von Raymond (1999) als Basar bezeichnet wird, entspricht aber durchaus auch den Prinzipien eines freien Marktes, welcher sich aufgrund von Angebot und Nachfrage wie von einer „unsichtbaren Hand“ geführt entwickelt (Smith 1974).

In der gesamten Diskussion um das Open-Source-Entwicklungsmodell wird immer noch weitgehend von unabhängigen Individuen ausgegangen. Wie bereits erwähnt, agieren jedoch bei den großen Projekten die Individuen nicht so selbstständig, sondern wirken im Auftrag ihres Arbeitgebers. Die Interviewergebnisse zeigen auf, dass der weitaus größte Teil der Firmenbeiträge nicht auf individueller Initiative basiert, sondern aus einer internen Projektplanung stammt. Diese setzt sich mehrheitlich zusammen aus internen Anforderungen (z. B. der Unterstützung eigener neuer Hardware oder Softwareversionen) sowie Wünschen bzw. Fehlermeldungen von Kunden und Partnern. Die Entwicklung ist folglich nicht primär von den Bedürfnissen des Entwicklers getrieben, sondern von den Geschäftsinteressen des Arbeitgebers. Es besteht jedoch durchaus auch Spielraum für die Auswahl und Bestimmung der konkreten Arbeiten durch die Entwickler.¹⁷¹

¹⁷¹ In den untersuchten Firmen ist es üblich, dass die von den Entwicklern gewünschten Beiträge in einer Planung erfasst und vom (Produkt-)Management abgesegnet werden müssen. Eine formelle Regelung wie sie z. B. Google hat, dass 10 % der Tätigkeit zur freien Mitarbeit in Open-Source-Projekten zur Verfügung stehen, wiesen die Firmen nicht auf.

Das Ausmaß der Selbstbestimmung der Codebeiträge der einzelnen Entwickler gegenüber konkreten Aufträgen ist dabei stark abhängig von der strategischen Ausrichtung der jeweiligen Firma in Bezug auf Open-Source-Software, wie sie in Kapitel 3.4.3. beschrieben wurde. Firmen, die sich strategisch kaum mit Open-Source-Software identifizieren, geben den Entwicklern eher mehr Freiheiten bezüglich der Zusammenarbeit mit der Community. Firmen hingegen, die eine klare Open-Source-Strategie aufweisen, haben diesbezüglich die Steuerung übernommen. Dieser Tatsache sind sich die Interviewteilnehmenden aller Typen bewusst. Ein konkretes Thema ist die Transition hin zur strategischen Ausrichtung nur für den Sozialromantischen Hacker. Die beiden beschriebenen Perspektiven – Freiheit der Entwickler und Strategie, jeweils in Bezug auf Open Source – finden sich auch in anderen Studien: Ersteres wird durch Henkel (2009) als „champions of revealing“ beschrieben, Letzteres durch Dahlander und Wallin (2006) mit dem „man on the inside“.

Aus Sicht der FLOSS-Projekte ist es relevant, ob sich mehrheitlich Freiwillige oder bezahlte Entwickler beteiligen. Dies geht aus Studien über das Apache- (Roberts et al. 2006), das Gnome- (Dahlander und Wallin 2006) sowie das Eclipse-Projekt (Spaeth et al. 2008) hervor. Festgestellt wurde, dass die durch ihren Arbeitgeber bezahlten Entwickler aktiver sind. Diese Tatsache wird auch durch den Linux-Kernel bestätigt. Mit durchschnittlich über 1.000 geänderten Codezeilen pro kommerziellen Entwickler gegenüber rund 650 im öffentlichen sowie privaten Umfeld ist ein markanter Unterschied festzustellen.

Die Firmenbeteiligung beeinflusst auch das Entwicklungsmodell, insbesondere in Bezug auf die Organisation und Koordination sowie auf die Rollen innerhalb der Projekte. Das Open-Source-Modell, wie es bisher in der Literatur diskutiert wurde, fügt sich diesbezüglich gut in aktuelle ökonomische Trends¹⁷² ein. Wie bereits in Kapitel 9.1.3 ausgeführt, sollte es somit nicht als Sonderfall verstanden werden. Das Basar-Modell muss folgerichtig aufgrund der starken Firmenbeteiligung zumindest partiell angepasst und die Selbstorganisation der Community hinterfragt werden. Dabei gilt es, die Rollen zu berücksichtigen, die aus der Mitarbeit von angestellten Softwareentwicklern in der Open-Source-Community entstehen. Deren Motivation, sich im Open- statt im Closed-Source- Umfeld zu betätigen, sowie die aus dem Umfeld resultierenden Konflikte sind ebenfalls dem Modell hinzuzufügen. Diese Diskussion

¹⁷² Diese Trends wurden im vorherigen Kapitel beschrieben. Einerseits ist eine zunehmende Dezentralisierung, andererseits ein Abbau von Eigentum festzustellen.

wird in den folgenden Kapiteln geführt.

9.2.1 Motivation von kommerziellen FLOSS-Entwicklern

Dass FLOSS-Entwicklung mehr Spaß macht, hat u. a. die Studie von Luthiger Stoll (2006) belegt. Der Autor beschränkt sich dabei auf die Unterscheidung von freiwillig mitarbeitenden FLOSS-Entwicklern und angestellten Closed-Source-Entwicklern. Die Gruppe der bezahlten FLOSS-Entwickler wird von Luthiger Stoll jedoch nicht thematisiert.

Die Entwicklung von Software macht Spaß, unabhängig vom Modell. Dies hat sich in den Interviewergebnissen dieser Studie bestätigt. Die meisten Teilnehmenden meinen, dass sie am liebsten während der gesamten Arbeitszeit Software entwickeln möchten. Dies trifft insbesondere auf den Typ des Pragmatischen Ingenieurs zu. Die meisten Personen, die firmenintern vom proprietären Bereich zu Linux gewechselt sind und somit einen direkten Vergleich anstellen können, empfinden die Open-Source-Arbeit eindeutig als „spaßiger“, ohne dies genauer spezifizieren zu können. Die Tatsache jedoch, dass sich ein Typ des Hedonistischen Programmierers in den empirischen Daten dieser Studie nicht bestätigen lässt, weist darauf hin, dass der Spaß an der Arbeit weniger als Motivator denn als Hygienefaktor gesehen werden kann. Die Unterscheidung von Motivatoren und Hygienefaktoren geht auf die Zwei-Faktoren-Theorie von Herzberg (1959) zurück. Unter den Hygienefaktoren versteht man diejenigen Gegebenheiten, die vorausgesetzt werden und deshalb nicht zur Zufriedenheit beitragen. Fehlen sie jedoch, so können sie zu Unzufriedenheit führen. Die Motivatoren wirken umgekehrt: Sind sie anzutreffen, so führen sie zur Zufriedenheit. Fehlen sie hingegen, so muss daraus nicht eine Unzufriedenheit resultieren.

Die Interviewergebnisse haben gezeigt, dass sich Open-Source-Entwickler nicht ausschließlich aus der Community rekrutieren lassen, sondern dass der firmeninterne Weg ebenso praktikabel ist (siehe dazu den soziodemografischen Überblick in Tabelle 8). Dies verändert auch die Motivationsdiskussion erheblich. Ein zentraler Aspekt, weshalb sich Softwareentwickler mit langjähriger Erfahrung im Closed-Source-Bereich zu Open Source im Allgemeinen und dem Linux-Kernel im Speziellen hingezogen fühlen, ist die Qualität des Codes. Dabei verstehen die Entwickler unter Qualität nicht primär die Stabilität und relative Fehlerfreiheit des Betriebssystems – also das, was den Anwender in erster Linie interessiert. Bedeutender sind das

Design und die Architektur sowie die Klarheit und Schönheit des Codes an sich, erwirkt durch klar definierte und auch konsequent durchgesetzte Codier-Standards. Hier manifestiert sich das Berufsethos der Betroffenen deutlich.

Die Motivationsdiskussion zeigt, dass Open Source und der Linux-Kernel nicht chaotisch und vorwiegend informell organisiert sind. Die Tatsache, dass es nicht nur geschriebene und ungeschriebene Regeln gibt, sondern dass diese auch gegen individuelle Interessen durchgesetzt werden, ist nicht nur für die Qualität der Software, sondern auch für die Motivation zur Mitarbeit von entscheidender Bedeutung. Insbesondere in der Durchsetzung der Standards zeigt sich gemäß den interviewten Personen ein entscheidender Unterschied zur Closed-Source-Entwicklung, wo aufgrund von oft kurzfristigen Firmeninteressen definierte und formell vorgegebene Standards nicht berücksichtigt werden. Dieser Aspekt wurde bei der bisherigen Diskussion von FLOSS vernachlässigt. Auch hier zeigt sich, dass die Organisation der Projekte – Flexibilität, Dezentralisierung etc. – viel weniger als Besonderheit des Entwicklungsmodells gesehen werden darf. Die Codequalität und die Methoden, wie diese durchgesetzt wird, sollten dagegen vermehrt ins Zentrum der Diskussion rücken.

9.2.2 Kommerzielle Rollen in FLOSS-Projekten

9.2.2.1 Einordnung der Typologie in die Community-Struktur

Wie bereits erwähnt, ist bei den angestellten FLOSS-Mitarbeitern von zwei Rollentypen auszugehen: „man on the inside“ (Dahlander und Wallin 2006) und „champion of revealing“ (Henkel 2009). Die Vertreter beider Rollenkonzepte sind dabei aus einer Firmensicht motiviert. Während dem „man on the inside“ die These zugrunde liegt, dass Firmen den Kontrollverlust durch den fehlenden Eigentumsanspruch durch Einflussnahme mittels eines gesponserten Kernentwicklers zumindest teilweise wieder zurückgewinnen können, basiert der „champion of revealing“ auf der Prinzipal-Agenten-Theorie und der (prinzipiell nicht bestätigten) Vermutung, dass die angestellten Entwickler die Interessen der Community höher halten als diejenigen der Firma. Während in beiden Rollentypen implizit von einer Entweder-

oder-Umschreibung ausgegangen wird, finden sich die beiden Typen in den in dieser Studie untersuchten großen IKT-Unternehmen gleichzeitig nebeneinander, sowohl innerhalb der Firmen als auch teilweise in Personalunion.

Die Rollen von Dahlander und Wallin sowie Henkel einerseits und die in dieser Studie erstellte Typologie andererseits kann wie in Abbildung 34 aufgezeigt in das Rollenmodell von Nakakoji et al. (2002, siehe dazu Kapitel 3.2.1) integriert werden.

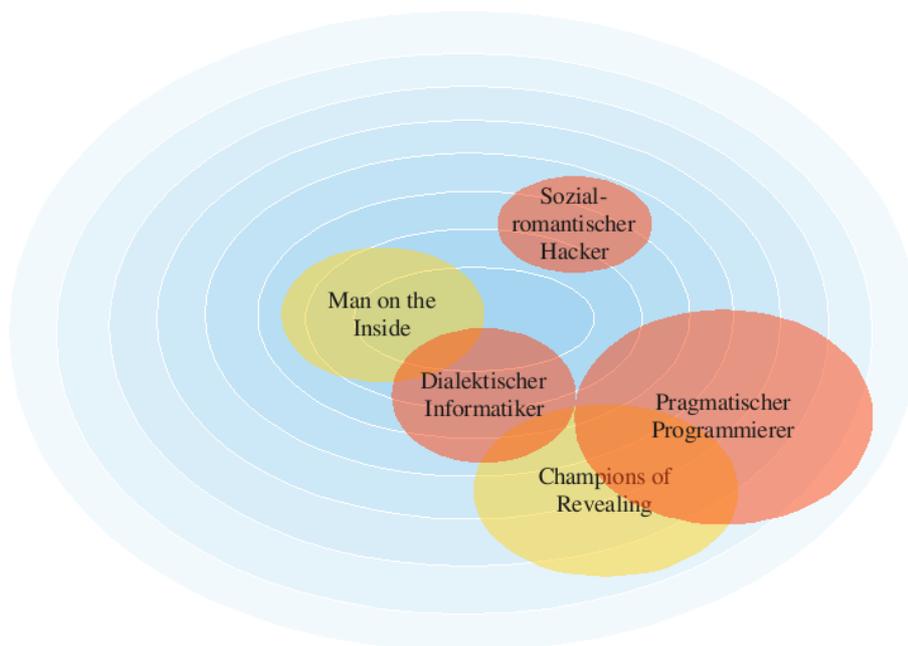


Abbildung 34: Rollen und Typologie von kommerziellen FLOSS-Entwicklern

Basierend auf den Interviewergebnissen kann die Rolle des „man on the inside“ lediglich zwei Personen zugeordnet werden. Beide entsprechen dabei dem Typ des Dialektischen Informatikers. Einer von ihnen wurde tatsächlich aus der Community rekrutiert, der andere hat sich diese Position aufgrund seiner Tätigkeit in der Firma erwerben können, wobei er „nur“ an einem Unterprojekt arbeitet. Eigen ist beiden Rollen, dass sie intern einen sehr großen technischen Entscheidungsspielraum haben und sie auch Arbeitszeit für die Erfüllung ihrer Community-Aufgaben nutzen können. Dies bedeutet allerdings, dass sie selbst gerade dadurch und aufgrund vermehrter administrativer Aufgaben und Management-Tätigkeiten nur noch wenig Code schreiben können. Aufgrund ihres Netzwerkes und der strategischen Bedeutung sind diese Angestellten nur schwer austauschbar und erhalten dadurch innerhalb der Firma eine gewisse Machtposition. Eine besondere Position nehmen dabei diejenigen Entwickler ein, die

hierarchisch direkt unter Linus Torvalds eingeordnet sind und die fast ausnahmslos von Firmen angestellt wurden. Zu ihnen gehören Leute wie Alan Cox (Red Hat), Greg Kroah-Hartman (Novell/Suse) oder Andrew Morton (Google). Sie können ihre Community-Aufgaben weitgehend unabhängig von ihrem Arbeitgeber erfüllen und wurden hauptsächlich angestellt, um für die Firma wichtige Projekte voranzutreiben, die ohne ihre Mitwirkung vermutlich nicht oder nur langsam vorangetrieben würden. Zudem wirkt sich ihre Anstellung förderlich auf die Reputation der jeweiligen Firma aus.

Die Dialektischen Informatiker unterscheiden sich in einem wichtigen Punkt vom Rollentyp „man on the inside“. Sie wirken nicht nur strategisch als Kontrollorgan der Firma, sondern dialektisch in einem modernen Sinne, versuchen also die Gegensätze von Community und Firma zu eruieren und aufzuheben. Ihre Bedeutung geht weit über das Verständnis von Dahlander und Wallin hinaus, indem sie nicht nur versuchen, in einem möglichen Rahmen Kontrolle über das Projekt zu gewinnen, sondern sowohl in der Firma als auch in der Community Einfluss nehmen. So können beide noch näher zusammenwachsen. Sie agieren zwar durchaus im Interesse ihres Arbeitgebers, ihr Wirken ist allerdings nachhaltig und kann somit über der Firma angeordnet werden.

Mehr Code als der Dialektische Informatiker können die „champions of revealing“ dem Kernel beisteuern. Henkel (2009) versteht darunter diejenigen Entwickler, die in der Firma zumindest inoffiziell entscheiden können, welcher Code in ein Open-Source-Projekt einfließen darf. Diese Position kann innerhalb der Firma als Vertrauensbeweis gelten, da Fehlentscheidungen aufgrund der Lizenzbedingungen kaum wieder rückgängig gemacht oder korrigiert werden können. Diese Rolle muss sich in der Community nicht zwingend widerspiegeln, setzt jedoch voraus, dass neben einem loyalen Firmenverhalten auch eine Vertrautheit mit der FLOSS-Community vorhanden ist. Henkel sieht bei diesen Personen ein schwieriges, wenn auch nicht unlösbares Prinzipal-Agenten-Problem, da sie sich oft sowohl der Firma als auch der Community verpflichtet fühlen und somit das Risiko besteht, dass sie die Firmeninteressen hinter diejenigen der Community stellen (siehe dazu auch Rossi und Bonaccorsi 2006). Die Rolle findet sich in den geführten Interviews in den Typen des Dialektischen Informatikers sowie des Pragmatischen Ingenieurs wieder. Es handelt sich dabei meist um Personen, die intern eine Teamlead-Funktion innehaben, welche mit den „lieutenants“ (Moody 2001) der Community verglichen werden können. Nicht selten haben sie gleichzeitig auch

eine Package-Verantwortung im Linux-Kernel, wobei es sich dabei in aller Regel um einen Bereich handelt, der hauptsächlich in der eigenen Firma betreut wird (wie z. B. die Unterstützung der eigenen Hardware). Zwar muss der Arbeitgeber auf die Loyalität seines Angestellten vertrauen, dieser wiederum hat eine zu wenig starke Position (sowohl in der Community wie in der Firma), um intern Macht ausüben zu können. Die stärkere Einbindung in die unternehmerische Verantwortung durch die Teamlead-Funktion könnte ein bewusster Weg der Firmen sein, um den Prinzipal-Agenten-Konflikt zugunsten des eigenen Betriebs zu beeinflussen.

Aufgrund der oben aufgeführten Erklärungen dürfte klar geworden sein, dass der Dialektische Informatiker weit über die Bedeutung der „champions of revealing“ hinausgeht. Der Pragmatische Ingenieur entspricht wohl am ehesten in der fortgeschrittenen Phase, also im Übergang zum Dialektischen Informatiker, der von Henkel beschriebenen Rolle. Ansonsten wird der Pragmatische Ingenieur, so wie er hier als „einfacher“ Entwickler porträtiert wird, in der Diskussion weitgehend vernachlässigt. Dies kann einerseits darauf beruhen, dass diese als weniger interessant angesehen werden. Andererseits ist denkbar, dass sie wegen ihrer geringen Aktivitäten in der Community – mit Ausnahme der Codebeiträge – nur wenig aktiv sind und nicht auf Online-Anfragen reagieren. Da in der vorliegenden Studie über die Firmen rekrutiert wurde, konnte diese Gruppe spezifisch angesprochen werden. Es darf vermutet werden, dass die Gruppe, die sich aus dem Typ des Pragmatischen Ingenieurs rekrutiert, quantitativ den größten Anteil an bezahlten Entwicklern ausmacht, wenn auch das gewählte Vorgehen keinen verlässlichen Schluss zulässt. Da sie im Netzwerk nicht sonderlich aktiv sind, sich weniger mit der Philosophie der FLOSS-Bewegung identifizieren und vorwiegend die Firmeninteressen vertreten, haben sie weder strategische Relevanz noch bieten sie Gefahr für einen Prinzipal-Agenten-Konflikt. Diese Personen sind austauschbar und definieren sich vorwiegend über ihre Programmierfähigkeiten. Für die Unternehmen sind sie jedoch enorm wichtig, da sie einen großen Teil des Codes schreiben, der für das eigene Geschäftsmodell von Bedeutung ist.

Wie Henkel in seinen Schlussfolgerungen erwähnt, ist ein Prinzipal-Agenten-Konflikt allein durch eine Vertrautheit mit der Community noch nicht gegeben. Erst durch die Identifikation mit der Ideologie von FLOSS ergeben sich gemäß Henkel diesbezüglich Probleme für die Firmen. Diese Aussage lässt darauf schließen, dass der Typ des Sozialromantischen Hackers dem „champion of revealing“ weitgehend entspricht und sich bei ihm erhebliche Konflikte er-

geben müssten. Diese Annahme kann jedoch nicht bestätigt werden. Der eine in dieser Arbeit untersuchte Fall hat die konfliktbeladenen Situationen weitgehend in sein Privatleben verschoben. Aufgrund seiner Aussagen wird vermutet, dass seine Firma die interne Organisation bereits so weit angepasst hat, dass gar keine Prinzipal-Agenten-Konflikte auftreten können.¹⁷³

Die in dieser Studie erarbeitete Typologie fasst die in der bisherigen wissenschaftlichen Auseinandersetzung erarbeiteten (strategischen) Rollen von FLOSS-Entwicklern im kommerziellen Umfeld nicht nur zusammen, sondern erweitert sie einerseits um den pragmatischen Aspekt. Andererseits kann durch die Typologie ein umfassendes Abbild der kommerziellen Arbeit am Linux-Kernel dargestellt werden. Die Ergebnisse der vorliegenden Studie legen zudem nahe, dass die Firmen weit weniger als vielerorts angenommen über Personalpolitik strategisch innerhalb der FLOSS-Community wirken. Die Tatsache, dass der Linux-Kernel und andere Projekte in erster Linie durch den geschriebenen Code vorangetrieben werden, spiegelt sich in den untersuchten Firmen wider. Es wird vornehmlich durch die Menge und die Art des beigetragenen Codes Einfluss genommen. Deshalb darf auch die Rolle des Pragmatischen Programmierers nicht unterschätzt werden, da seine Beiträge weitgehend durch die interne Planung bestimmt werden und somit in weiten Teilen der bekannten proprietären Softwareentwicklung entsprechen. Trotzdem bedarf es auch des Dialektischen Informatikers, gerade weil sich der Pragmatische Programmierer nur bedingt in der Community zu Hause fühlt und so die notwendige Unterstützung im Sinne von flankierenden Maßnahmen bekommt.

9.2.2.2 Wechselwirkungen bei den Typen

Die in der Konklusion hergeleitete Grafik zu den Wechselwirkungen bei der Beteiligung an Open Source kann aufgrund der Interviewergebnisse und unter Berücksichtigung der drei definierten Typen wie in Abbildung 35 dargestellt angepasst werden.

¹⁷³ Dass dies nicht gerade zum Vorteil des Angestellten vom Typ des Sozialromantischen Hackers ist, hat das Beispiel des Prototyps, wie unter Kapitel 8.5.1 beschrieben, klar aufzeigen können.

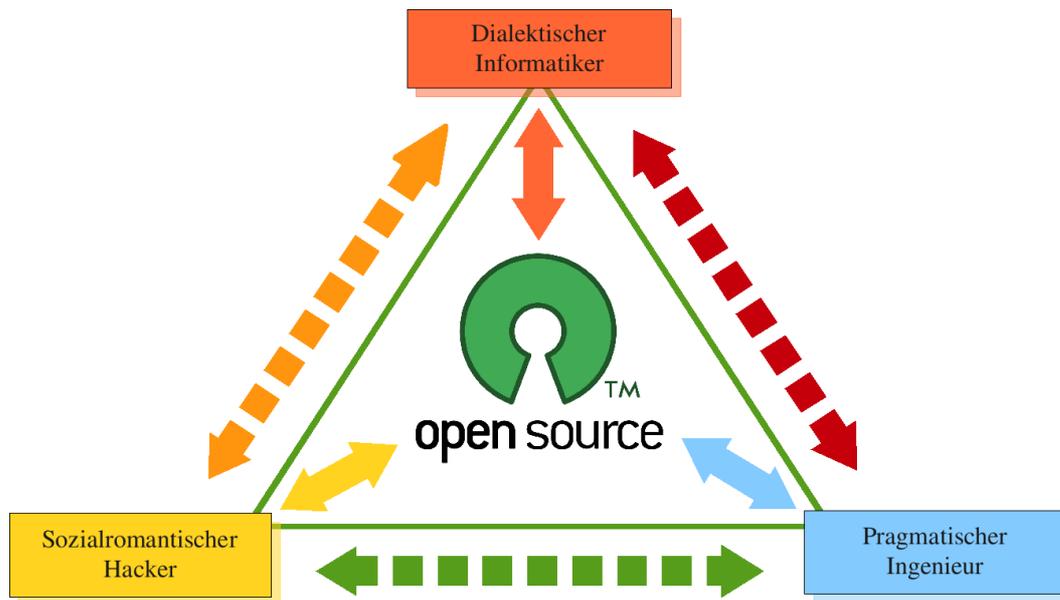


Abbildung 35: Wechselwirkungen zwischen den Typen

Der Sozialromantische Hacker wirkt insofern auf die Open-Source-Community, als er die der Bewegung zugrunde liegende Ideologie weiterträgt und so dafür sorgt, dass die Community integer bleibt. Sollte sich die Community von der Ideologie wegbewegen, so dürfte für den Sozialromantischen Hacker die Mitarbeit uninteressant werden. Er sorgt jedoch auch dafür, dass, wie von Rossi und Bonaccorsi (2006) festgestellt (und im Kapitel 3.4.4 bereits zitiert), die soziale Komponente von Open Source in die Firma getragen wird. Wie auch Henkel (2009) konstatiert hat, ist dies jedoch der Ausnahmefall und eher in kleinen und mittleren Open-Source-Firmen anzutreffen. In großen Firmen, so legen es auch die Interviewergebnisse nahe, ist der Einfluss der Sozialromantischen Hacker marginal.

Vom Pragmatischen Ingenieur wird lediglich über den geschriebenen Code direkt Einfluss auf Open Source genommen, da er sich ansonsten wenig in der Community beteiligt und auch die Ideologie nicht teilt. Über seine Codebeiträge und das sachliche, professionelle Verhalten hat er dennoch einen nicht zu unterschätzenden Einfluss. Denn wie das evolutionäre Entscheidungsmodell (siehe Abbildung 30) erklärt, wirkt ein aktuelles Verhalten über den Feedback-Loop auf die zukünftigen Werte und Normen der Community. Für die Firma ist der Pragmatische Ingenieur ein ganz normaler Softwareentwickler, wie er auch in Closed-Source-Projekten anzutreffen ist. Den Sozialromantischen Hacker beachtet er wenig, da er ihm weitgehend fremd ist. Wenn sich die Community durch den Einfluss der Pragmatischen Ingenieure weg

von der Ideologie bewegt, wendet sich der Sozialromantische Hacker von Open Source ab bzw. sucht eine Community, die mehr seiner Haltung entspricht.

Der Dialektische Informatiker schließlich wirkt als Vermittler zwischen Community und Firma und insofern auch zwischen Community und Pragmatischem Ingenieur bzw. zwischen Firma und Sozialromantischem Hacker. Dabei verändert er sowohl die Community als auch die Firma und beeinflusst den Pragmatischen Ingenieur direkt, den Sozialromantischen Hacker eher indirekt.

Während Dahlander und Wallin (2006) sowie Henkel (2009) die direkte Einflussnahme durch die bezahlten Entwickler auf die Community thematisieren, sind es bei Dahlander und Magnusson (2006) sowie Rossi und Bonaccorsi (2006) eher indirekte Einflüsse auf die Community und Firmen über die Entwickler. Was jedoch beide nicht ansprechen, ist, dass sich durch die Einflussnahme das ganze System verändert und damit wiederum Einfluss auf andere Beteiligte genommen wird. Das evolutionäre Entscheidungsmodell legt demnach nahe, dass die Strategie und das daraus abgeleitete Verhalten der Firmen einen nachhaltigen Einfluss auf die Normen und Werte der Community haben. Wie verschiedene Studien belegen (West und O'Mahony 2005; Dahlander und Magnusson 2006; Spaeth et al. 2008), haben Open-Source-Projekte, die zu stark durch eine Firma dominiert werden, Probleme dabei, freiwillige Mitstreiter zu gewinnen. Trotz der dominierenden Firmenbeteiligung im Linux-Kernel ist es dennoch eine der Stärken des Projekts, dass immer noch zahlreiche Freiwillige in ihrer Freizeit das Projekt unterstützen. Einerseits werden so reale Probleme rasch gelöst, die in einem kommerziellen Umfeld weniger beachtet werden. Andererseits kann nur so ein Wissenspool angesprochen werden, der ansonsten den Projekten verwehrt bliebe.

9.2.2.3 Der Umgang der verschiedenen Entwicklertypen mit Konflikten

Open-Source-Entwickler im kommerziellen Umfeld sehen sich mehreren spezifischen Problemen gegenüber, die direkten Einfluss auf ihre Arbeitsgestaltung haben. Diese Probleme werden unterschiedlich wahrgenommen, und die verschiedenen Typen haben unterschiedliche Strategien zu deren Bewältigung entwickelt. Konkret wurden in den Interviews vier Problembereiche thematisiert (siehe Tabelle 10).

Komplexe Planung
Legale Einschränkungen
Interessenkonflikte
Kommunikation mit der Community

Tabelle 10: Problembereiche von Open-Source-Entwicklern in großen IT-Firmen

Komplexe Planung: Der Planungsprozess im Linux-Umfeld ist äußerst komplex, da der Ersteller des Codes meist nicht der Vertreiber (Distributor) ist.¹⁷⁴ Daraus ergibt sich, dass die Distributoren interne Pläne von mehreren unabhängig agierenden Partnern vereinbaren müssen, ohne dass alle Beteiligten an einem Tisch sitzen.¹⁷⁵ Daraus resultiert einerseits, dass Software- und Hardwarehersteller nur ein oder maximal zwei Distributionen unterstützen.¹⁷⁶ Andererseits haben sich die beiden größten Distributoren, Red Hat und Suse, informell so arrangiert, dass sie ihre Releases jeweils zeitverschoben auf den Markt bringen.¹⁷⁷ Als Konsequenz hat in den Firmen, die in größerem Umfang Linux mit Arbeitskraft unterstützen, eine Aufgabenteilung stattgefunden. Der Entwickler wird dabei weitgehend von der konkreten Terminkoordination entbunden, da dies im Zuständigkeitsbereich von Planern und Koordinatoren liegt. In letzter Instanz spürt der Entwickler trotzdem Terminkonflikte und muss sie in seiner Arbeit auch lösen. Die Verantwortung dafür trägt er aber nicht.

Während der Pragmatische Ingenieur versucht, Probleme mit Terminen nach oben zu delegieren, nimmt der Dialektische Informatiker oft eine vermittelnde Position ein und sucht nach konkreten Lösungen. Dies ist auch eine Folge davon, dass er in der Regel in seiner firmeninternen Position mehr diesbezügliche Verantwortung hat. Festzustellen ist, dass beide Typen die Planung als ein notwendiges Übel wahrnehmen, dafür aber eine befriedigende Handlungs-

174 IBM z. B. hat sich im Gegensatz zu Oracle aufgrund strategischer Überlegungen explizit dagegen entschieden, eine eigene Linux-Distribution ins Portfolio zu übernehmen.

175 Eine detailliertere Beschreibung dazu folgt weiter unten.

176 Dies führt wiederum zu einem Lock-in, der mit FLOSS beseitigt werden soll. Da sich dies jedoch nur auf die Distribution bezieht, die in der Regel kostenlos ist, findet über den Supportbereich trotzdem ein, wenn auch reduzierter, Wettbewerb statt. Hingegen ist es für eine neue Distribution sehr schwierig, im Geschäftsbereich Fuß zu fassen – ein Umstand, den auch die mit einem sehr großen finanziellen Background versehene Distribution Ubuntu zu spüren bekommt. Diese weist zwar eine enorme Verbreitung vor, wird aber auf den Servern oft noch nicht vom Hardwareanbieter zertifiziert.

177 Dieses Arrangement kam nicht ganz freiwillig zustande. Wenn beide Distributionen etwa zur selben Zeit einen Major-Release planen, bedeutet dies für die Partner, die vertragliche Abmachungen zu beiden Distributionen aufweisen, dass sie zur selben Zeit den Hauptaufwand betreiben müssen. Dies hat in der Folge immer wieder zu Terminverschiebungen geführt, worauf die betroffenen Partner Druck auf die Distributionen ausgeübt hatten, dass sie ihre Releases zeitverschoben planen müssen.

alternative gefunden haben. Der Sozialromantische Hacker schließlich interessiert sich weniger für firmeninterne Termine. Dass er trotzdem in die interne Planung eingebunden ist und dadurch Qualitätseinbußen akzeptieren und damit eine Community-Regel brechen muss, führt bei ihm zu einer inneren Spannung.

Legale Einschränkungen: Alle im Setting vorhandenen Firmen haben sich zu Open-Source-Software verpflichtet, jedoch in unterschiedlichem Maße. Daraus ergibt sich, dass die Bereitschaft, Code offenzulegen und freizugeben, noch unterschiedlich stark ausgeprägt ist. Als Folge davon sind auch die Regeln und Prozesse für die Freigabe von Code noch sehr unterschiedlich weit fortgeschritten. Es kann jedoch diesbezüglich in jedem Fall eine Entwicklung zur Standardisierung verfolgt werden.

Konkret sind die Firmen sehr darauf bedacht, keine nur für den internen Gebrauch vorgesehenen Informationen preiszugeben. Bei den Hardwareherstellern sind das zum Beispiel die Spezifikationen der Hardware, bei Softwareproduzenten die konkreten Algorithmen. Für die Entwickler bedeutet dies, dass sie in Eigenverantwortung ihren Code diesbezüglich überprüfen müssen, da es keine Instanz gibt, die geschriebenen Code vor der Herausgabe nochmals überprüft. Im Zweifelsfalle besteht die Möglichkeit, bei der juristischen Abteilung nachzufragen.

Eine weitere legale Beschränkung ist, dass sich die Firmen das Recht vorbehalten, Funktionalität zu patentieren oder verschlossen zu halten, falls dies aus unternehmerischer Sicht als angebracht erscheint. Dieses Problem reduziert sich jedoch in der Praxis sehr, weil dies nur bei neu zu schreibender Funktionalität zum Tragen kommt. Die tägliche Arbeit besteht zum größten Teil darin, bestehenden Code anzupassen oder zu korrigieren.

Während die oben beschriebenen Punkte durch alle drei Typen wahrgenommen, aber nicht als konkretes Hindernis in der Arbeit gesehen werden, sieht es bei haftungsrelevanten Gesichtspunkten anders aus. Der Pragmatische Ingenieur sieht sich durch seinen Arbeitsvertrag von der Haftung entbunden und thematisiert dies nicht weiter. Der Dialektische Informatiker hingegen sieht sich in seiner Arbeit behindert, da er nicht den Community-Prozessen entsprechend agieren kann.

Der Sozialromantische Hacker positioniert sich bei legalen Fragen auf der Seite der Free Software Foundation. Patente möchte er gerne abschaffen und am liebsten sämtliche Informationen frei zugänglich machen. In der täglichen Arbeit kann er sich weitgehend arrangieren, er

bekundet jedoch starke Probleme mit der Tatsache, dass seine Firma in der Regel auf einer gegenteiligen Position beharrt.

Interessenkonflikte: Die Vertretung von Firmeninteressen in der Community ist unbeliebt und eher kontraproduktiv, vor allem wenn es darum geht, Code in der gewünschten Form in den Kernel zu bekommen. Die Linux-Kernel-Community ist tatsächlich weitgehend immun gegen kommerzielle Argumente. Es zählen ausschließlich technische Aspekte. Durch die Übernahme einer Maintainerfunktion in der Community kann jedoch durchaus Einfluss im Sinne der Firma genommen werden, was aus den Interviewergebnissen auch hervorgeht.

Der Pragmatische Ingenieur vertritt die Interessen seines Arbeitgebers, ist sich jedoch bewusst, dass er diese Interessenvertretung in der Community nicht bekannt geben kann. Er sieht seine Möglichkeit zur Interessenvertretung deshalb hauptsächlich darin, hartnäckig zu sein, wenn seine Vorschläge nicht zeitnah akzeptiert werden, und sachlich zu argumentieren, auch wenn er persönliche Kommentare erhält. Da er die Konflikte, die aus dieser Haltung entstehen, nicht selbst austragen muss, sind sie für ihn kaum bedeutsam. Durch sein vermittelndes und strategisches Denken versucht der Dialektische Informatiker, Interessenkonflikte zu vermeiden. Er versteht die Zusammenarbeit mit der Community auch als Investition in eine Machtposition. Aufgrund des meritokratischen Charakters der Gemeinschaft über vergangene Leistungen können die zukünftigen Interessen stärker vertreten werden. Der Sozialromantische Hacker schließlich vertritt die Interessen der Open-Source-Community und hat Probleme damit, Firmeninteressen zu akzeptieren, die seiner persönlichen Überzeugung widersprechen.

Kommunikation mit der Community: Die Kommunikation wird von den meisten Interviewteilnehmenden als einer der großen Unterschiede zwischen Open-Source- und Closed-Source-Software gesehen. Einerseits ist darauf zu achten, was an Informationen, die nur für internen Gebrauch gedacht sind, preisgegeben wird. Zudem ist den Entwicklern auch bewusst, dass sie ihre Firma in der Community vertreten und deshalb ein professioneller Stil gefordert ist. Andererseits müssen die befragten Personen akzeptieren können, dass Entscheidungen nicht aufgrund von Machtpositionen getroffen, sondern sachlich ausdiskutiert werden.¹⁷⁸

178 Die Kommunikation in der Community entspricht somit eher einer basisdemokratischen Diskussion als einem hierarchischen Entscheidungsprozess. Die Entwickler stehen dabei oft in dem Konflikt, dass sie firmenintern hierarchisch gefällte Entscheidungen auf dem Diskussionsweg in die Community tragen müssen.

Dass der Kommunikationsstil insbesondere in der Linux-Kernel-Community mitunter persönlich ausfallend wird, empfindet der Pragmatische Ingenieur als störend und hemmend. Er würde den Entscheidungsprozess gern durch ein Machtwort verkürzen, akzeptiert jedoch die Spielregeln, die in seinen Augen Eigenschaften wie „Offenheit“, „Hartnäckigkeit“ und „ein dickes Fell“ erfordern. Für den Dialektischen Informatiker ist das Diskutieren sachlicher Entscheidungen eine Möglichkeit, vermittelnd zu wirken. Der schroffe Stil wird von ihm nicht weiter thematisiert, sondern er versucht, als Vorbild voranzugehen und persönliches Befinden von sachlichen Argumenten zu trennen. Der Sozialromantische Hacker schließlich fühlt sich in der Community wohl und hat keine Probleme mit eventuellen sprachlichen Auswüchsen.

Ferner sind interne Meetings ein wichtiges Thema in Bezug auf die Kommunikation. Meetings werden von allen Typen als störend und tendenziell als zu häufig angesehen. Der Pragmatische Ingenieur versucht, sich so weit wie möglich von formellen Meetings fernzuhalten, die oft mehr organisatorischen Belangen dienen als der Klärung konkreter Probleme. Bevorzugt werden von ihm ad-hoc-Sitzungen zu technischen Fragen. Bei den formellen Meetings verhält er sich eher passiv und empfindet sie deshalb als Einweg-Kommunikation. Für den Dialektischen Informatiker sind Meetings weitgehend Mittel zum Zweck, und er passt sich dabei den Standards der jeweiligen Gremien an (z. B. Powerpoint-Präsentationen im Management-Bereich). Er ist aktiv und versucht, konsensorientierte Lösungen zu finden. Der Sozialromantische Hacker schließlich sieht Zusammenkünfte eher als Diskussionsforum. Mit den hierarchischen Machtunterschieden in den Meetings hat er Mühe und verhält sich in ihnen eher passiv. Er bevorzugt es, in persönlichen Situationen oder über die etwas unpersönlichere schriftliche Variante zu kommunizieren.

9.2.2.4 Transitionen der Typen

Entgegen den Ergebnissen von Dahlander und Wallin (2006) sowie Henkel (2009) zeigen sich in den Interviewergebnissen Transitionen bei den Typen. Drei verschiedene Übergänge sind erkennbar: Erstens beim Eintritt des Arbeitnehmers in die bezahlte Linux-Tätigkeit; zweitens verändert sich die Arbeit im Linux-Umfeld innerhalb der Firma (siehe Kapitel 3.4.3), und drittens sieht sich auch der angestellte Linux-Entwickler selbst in einem Reifeprozess begriffen (siehe Abb. 36).

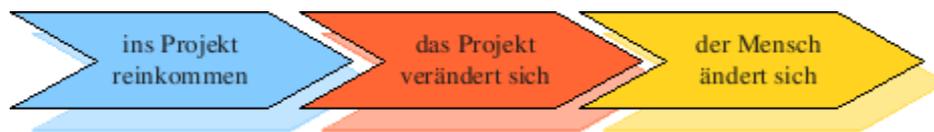


Abbildung 36: Transitionen der Linux-Tätigkeit in Firmen

Ins Projekt reinkommen: Eine Motivation für die Beteiligung an Open-Source-Software, die in der Literatur häufig erwähnt wird, sind zukünftige Karrierechancen (Lerner und Tirole 2001; Lakhani und Wolf 2003; Ghosh et al. 2002). Es ist deshalb naheliegend, dass Linux-Kernel-Entwickler aus der Community angeworben werden. Bei zwei der siebzehn Interviewteilnehmenden war dies tatsächlich der Fall, jedoch geschah dies bereits vor geraumer Zeit (1999 bzw. 2001). Während ein Teilnehmer in einer Massen-E-Mail angeschrieben wurde, erreichte den zweiten aufgrund seiner konkreten Erfahrung und Position in der Community – er ist Maintainer in einem für die anwerbende Firma wichtigen Bereich – eine individuelle Anfrage. Dass eine Anwerbung aus der Community in späteren Jahren bei den Interviewteilnehmenden nicht mehr erfolgte, könnte daran liegen, dass mittlerweile der größte Teil der Entwickler von einer Firma angestellt ist. Deshalb wäre es aufgrund von möglichen Gegenrekrutierungen sowie aus Konkurrenzgründen heikel, diese Mitarbeitenden direkt anzufragen.

Wenn die Entwickler vor ihrer Anstellung bereits bei einer anderen Firma tätig waren – immerhin beinahe ein Drittel der befragten Personen –, so hatten sie ohne Ausnahme bereits in ihrer vorherigen Tätigkeit mit Linux zu tun. Bei allen ist der Wechsel von den Arbeitnehmern ausgegangen. Es gibt folglich keine Anzeichen, dass Entwickler aufgrund ihrer Bekanntheit in der Community direkt angeworben wurden.

Somit bleiben noch zwei Einstiegsmöglichkeiten für das bezahlte Arbeiten an Linux: Einerseits wechselten vier Interviewteilnehmende innerhalb der Firma von einem Closed-Source-Projekt zu Linux, wobei die Initiative in allen vier Fällen vom Arbeitnehmer ausging. Andererseits gibt es die Möglichkeit, direkt nach einem Hochschulabschluss als Linux-Entwickler einzusteigen. Dies war bei über einem Drittel der Stichprobe der Fall, jedoch waren die Entwicklungsleistungen in der Community nur bei einer einzigen Person das maßgebliche Kriterium für die Einstellung. Trotzdem scheint es plausibel, dass sich eine Investition in Open-Source-Software während der Studienzeit durchaus bezahlt machen könnte. Diese Investition findet zur Zeit einerseits auf privater Basis statt, andererseits durch die Erstellung von Semes-

ter- und Abschlussarbeiten. Eine breite Integration des Themas Open Source in den Unterrichtsstoff lässt dies als gerechtfertigt erscheinen.

Das Projekt selbst verändert sich: Zu Beginn der Open-Source-Tätigkeit einer Firma sind Organisationsstruktur und Prozessabläufe noch nicht angepasst. Der entsprechende Bereich genießt weitgehende Freiheiten in der Arbeitsgestaltung, hat eine flache Hierarchie und ist im Allgemeinen sehr kollegial und von viel Idealismus geprägt. Die Hauptaufgabe der Linux-Kernel-Entwicklung besteht darin, erstmals den eigenen Geschäftsbereich in den Kern einzubringen und bedeutet demnach sehr viel Recherchearbeit.

Mit der zunehmenden Bedeutung von Open-Source-Software für die Unternehmen, einhergehend mit einer größeren Kundenbasis in diesem Bereich, werden auch die Prozesse standardisiert und der Bereich wird zunehmend strukturierter und hierarchischer organisiert. Dabei werden vorerst die in der Firma bewährten Prozesse übernommen und – soweit notwendig – an die Arbeit in der Community angepasst. Die Entwickler sehen sich zusehends mit Anforderungen von Produktmanagern und Kunden konfrontiert, für die primär Fehler zu beheben sind oder die neue Funktionalität wünschen. Die konkrete Tätigkeit der Linux-Kernel-Entwickler entspricht in dieser Form der klassischen Entwicklungstätigkeit, wie sie auch in der Closed-Source-Softwareentwicklung anzutreffen ist.

Mit zunehmender strategischer Relevanz sind die Unternehmen bereit, ihre internen Prozesse und Strukturen zu hinterfragen. Es findet auch hier eine Annäherung an die Open-Source-Community statt, jedoch immer mit dem Unternehmensziel im Fokus. In der Linux-Kernel-Entwicklung sind heute zunehmend strategisch denkende Informatiker gesucht, die es verstehen, die beiden Welten möglichst unvoreingenommen zusammenzubringen.

Der Mensch ändert sich: Die meisten der interviewten Personen waren zu Beginn ihrer Open-Source-Anstellung stark intrinsisch motiviert. Dies war in der Frühphase der Open-Source-Tätigkeit des Unternehmens kein Störfaktor, sondern sogar erwünscht, da nur mit dem nötigen Enthusiasmus die Open-Source-Bewegung in der Firma ins Rollen kam. Zunehmend wurde die Arbeit jedoch standardisiert und die firmenweiten Prozesse weitgehend integriert. Diesen Prozess haben viele Entwickler mitgemacht, was als normaler Entwicklungsprozess gesehen werden kann. Im Laufe der Jahre haben die Entwickler sich niedergelassen und eine Familie gegründet. Der Spaß an der Arbeit wurde ergänzt durch ein Bedürfnis nach Sicherheit

und ein ausreichend hohes Einkommen. Dieser Pragmatismus passte gut zur internen Professionalisierung der Open-Source-Entwicklung.

Einige der Personen, vornehmlich privat unabhängig, haben ihren Enthusiasmus behalten, konnten sich aber der zunehmenden Professionalisierung anpassen. Sie sehen ihre Aufgaben vor allem darin, die beiden Welten weiterhin näherzubringen.

9.2.3 Vertraglich abgesicherte Selbstorganisation

Die im Kapitel 3.2.1 beschriebenen vier Eigenschaften eines sich selbst organisierenden Systems – Komplexität, Selbstreferenz, Redundanz und Autonomie – treffen im Wesentlichen auf den Linux-Kernel zu. Während die Komplexität des Systems wohl unbestritten ist, müssen bei den anderen drei Eigenschaften aufgrund der Interviewergebnisse doch Einschränkungen gemacht werden. Diese beziehen sich weniger auf die technische Arbeit, sondern eher auf die Koordination. Diese Einschränkungen zeigen sich in der Form, wie sich die am Linux-Kernel beteiligten Firmen intern organisiert haben.

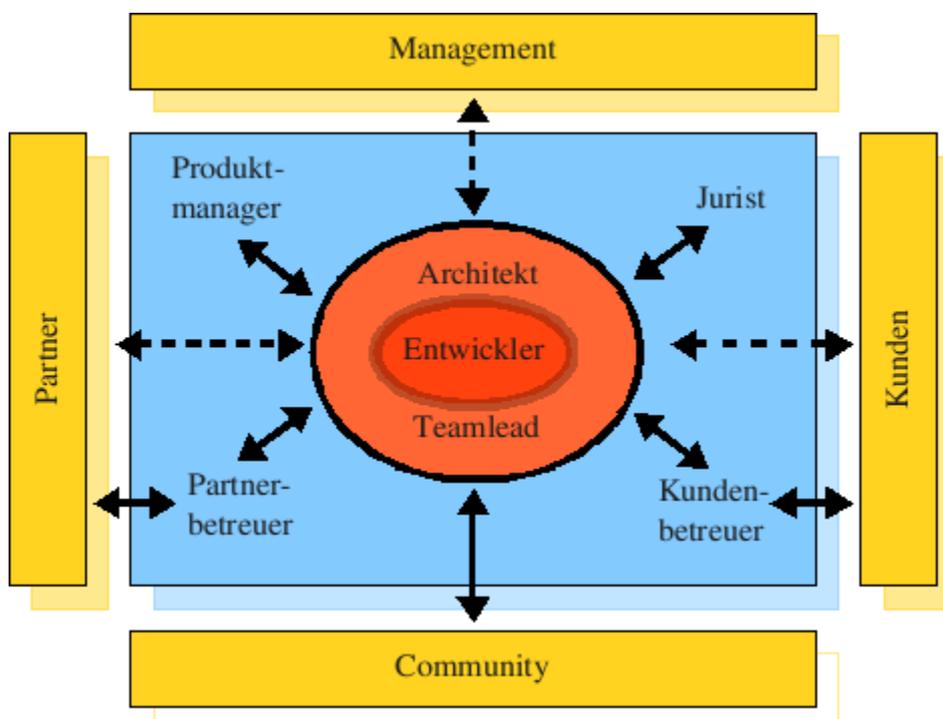


Abbildung 37: Organisation der Linux-Kernel-Entwicklung in den untersuchten Firmen

Wie aus der Abbildung 37 hervorgeht, sind die gestaltenden und organisierenden Rollen klar

getrennt. Der rote Teil ist als Bereich innerhalb der Firmen zu verstehen, der dem eigentlichen System des Linux-Kernels zuzurechnen ist. Hier wird vor allem gestaltet, aber in einem kleineren Umfang auch organisiert und geleitet. Die blau markierte Box beinhaltet firmeninterne Rollen, die nicht mehr direkt mit dem Linux-Kernel interagieren, sondern dies aufgrund von organisatorischen Regelungen zur Hauptsache über die Entwickler tun. Dabei gilt die Regel, dass Entwickler nur über technische Belange – sofern sie nicht geheim sind – reden dürfen. Alle planerischen und produktorientierten Prozesse müssen über diese Linie geschehen.¹⁷⁹ Bereits hier ist erkennbar, dass Selbstreferenz, Redundanz und Autonomie im Linux-Kernel nicht mehr vollständig gegeben sind.

Hinzu kommt, dass durch Partner und Kunden indirekt Einfluss genommen wird, also durch organisatorische Einheiten, die außerhalb der Firma liegen. Eine zentrale Rolle spielen dabei die Distributoren¹⁸⁰, die beim Linux-Kernel quasi als Schaltstelle aller Interessierten wirken. Der Distributor ist durch einzelne Verträge mit Partnern und Kunden verbunden. Interessant dabei ist, dass die Planung und Koordination nicht an einem runden Tisch stattfindet, sondern der Distributor die Planung mit jedem einzelnen Partner unabhängig voneinander macht, u. a. weil die vertraglichen Abmachungen nicht offengelegt werden. Aus Vertragssicht handelt es sich somit (siehe Abb. 38) um Zweierbeziehungen zwischen Distributor und Partner.¹⁸¹ Änderungen in der Planung – z. B. aufgrund einer Lieferverzögerung eines Partners – werden individuell mit allen anderen Partnern abgestimmt. Daraus resultieren möglicherweise neue Terminverschiebungen, die wiederum mit allen Partnern einzeln abgesprochen werden müssen. Deshalb ist das Management der Pläne für die betreffenden Stellen bei den Distributoren eine

179 Interessanterweise haben sich die untersuchten Firmen in einer ähnlichen Form organisiert, die sich vor allem in der Reife des Open-Source-Prozesses (siehe dazu Kapitel 3.4.3) herauskristallisiert. Eigen ist allen Firmen auch, dass sie den Linux-Bereich bis zu einem gewissen Grad isoliert haben. Zum einen hat dies legale Gründe: So kann auf einfachste Weise erreicht werden, dass kein proprietärer Code in ein FLOSS-Projekt einfließt und umgekehrt kein FLOSS-Code in ein proprietäres Projekt. Zum anderen ist es so möglich, die Regeln der Community-Arbeit in die eigene Organisation einfließen zu lassen, um so die Zusammenarbeit der Entwickler zu vereinfachen, ohne dass die gesamte Firma „infiltriert“ wird. Diese Praxis hat sich auch auf Anwenderseite bewährt, wie das Beispiel des deutschen Auswärtigen Amtes zeigt (Auener 2008).

180 Die Hard- und Softwarevendors liefern zwar in aller Regel ihren Code unabhängig von den Distributoren direkt ins Kernel-Repository. Da ihre Änderungen aber nur über die Distributoren zu ihren Kunden kommen und die Distributoren aus Stabilitätsgründen in ihren Produkten den Kernel nicht regelmäßig komplett aktualisieren, müssen Änderungen punktuell dort zusätzlich nachgeführt werden (sogenannter Backport), was sehr aufwendig ist.

181 Dies funktioniert auch, wenn zwei Partner eines Distributors wiederum vertraglich miteinander verbunden sind. Auch in diesem Falle werden keine Informationen über den Distributor in einem Dreiecksverhältnis ausgetauscht, sondern immer nur in der vertraglich abgesicherten Zweierbeziehung.

hochkomplexe Herausforderung.

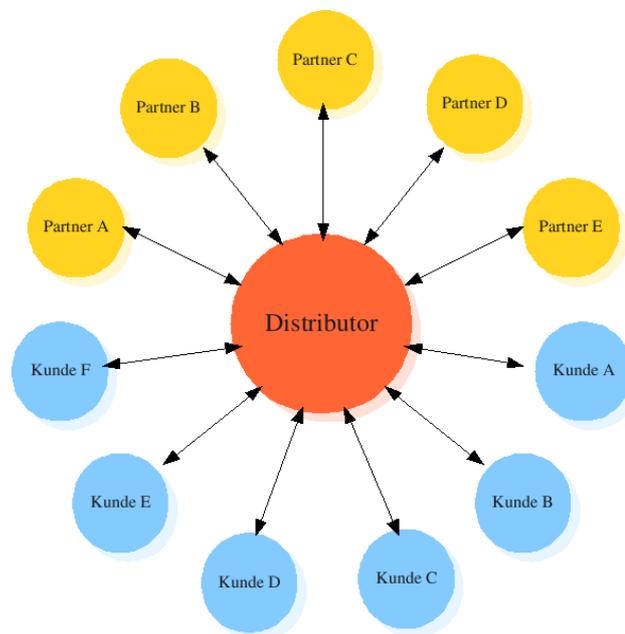


Abbildung 38: Koordination durch Distributoren

Aber auch die Hard- und Softwarevendors sind in der Planung gefordert. Als logische Konsequenz ergibt sich, dass diese meist nur zwei Distributionen zertifizieren. In der Regel sind dies Red Hat und Suse. Diese Reduktion geschieht, weil die Planung mit einer Distribution, wie soeben erläutert, sehr komplex ist. Die Komplexität steigt durch jede zusätzliche Distribution exponentiell an, da die Distributoren als Konkurrenten am Markt auftreten und deshalb auch keine Koordination untereinander stattfindet.¹⁸² Aufgrund der vertraglichen Abmachungen müssen einerseits die Distributoren aufpassen, dass sie keine Geschäftsinterna der Partner, die aufgrund eines NDAs (Non Disclosure Agreement) ausgetauscht werden, anderen Partnern vermitteln. Umgekehrt müssen die Hard- und Softwarevendors darauf achten, keine Interna des einen Distributoren an den anderen oder einen eigenen Kunden zu vermitteln. Auf diese Problematik haben die größeren Firmen reagiert, indem sie voneinander weitgehend unabhängige Koordinatoren und Teams bilden, die ausschließlich mit dem jeweiligen Partner, sei es Distributor, sei es Hard- bzw. Softwarevendor, zusammenarbeiten. Eine große Heraus-

¹⁸² Wie bereits früher erwähnt, ist es die Ausnahme, dass die beiden Distributoren informell ihre Release-Zyklen aufeinander abgestimmt haben, sodass diese zeitlich verschoben stattfinden, damit ihre Partner nicht in einen Prioritätskonflikt geraten.

forderung dabei ist, die verschiedenen Planungen¹⁸³ und Abhängigkeiten zu koordinieren und die eigenen Interessen durchzusetzen. Dazu ist einerseits ein großes Verhandlungsgeschick nötig, andererseits die Fähigkeit, sich in die Partnerseite einzufühlen, um die eigene Planung vorsorglich anpassen zu können.

Als Fazit hat sich herauskristallisiert, dass zwar das Linux-Kernel-Projekt im engeren Sinne durchaus als sich selbst organisierendes System bezeichnet werden kann. Betrachtet man das System jedoch ganzheitlicher einschließlich des Ökosystems mit Distributoren sowie Hard- und Softwarevendedoren, die zusammen einen Großteil des Sourcecodes beisteuern, so ist erkennbar, dass ein zusätzliches komplexes juristisches Netzwerk geknüpft wird, um die individuellen (kommerziellen) Interessen zu schützen.

183 Erschwerend kommt noch hinzu, dass die eigenen Planungen und diejenigen der Partner unterschiedliche Zeithorizonte haben können. Üblicherweise haben Hardwarevendedoren einen viel weiteren Planungshorizont als die Softwarevendedoren und die Distributoren. Zusätzlich planen größere Firmen in der Regel eher weiter im Voraus als kleine Unternehmen.

10 Schlussfolgerungen

Die Analyse der Linux-Kernel-Logdateien zeigt deutlich auf, dass die Weiterentwicklung des Linux-Kernels stark durch kommerzielle Interessen vorangetrieben wird. Dass diese Interessen durch zahlreiche und bezüglich Größe und Tätigkeitsbereich sehr unterschiedliche Firmen wahrgenommen werden, führt dazu, dass sich der Linux-Kernel in unterschiedlichste Richtungen weiterentwickelt. Der Einfluss einer einzigen Firma ist eingeschränkt, da sich breit gestreute Gruppierungen an der Entwicklung beteiligen. Deshalb gibt es keine einzelne Firma, die eine solch starke Position in der Community aufweist, dass sie entscheidend Einfluss nehmen kann. Für diese limitierten Einflussmöglichkeiten sorgt auch Linus Torvalds als „wohlwollender Diktator“.¹⁸⁴ Berücksichtigt man zudem, dass sich die „zweite Garde“ des Linux-Kernels fast durchweg in Positionen befindet, in der sie zwar von kommerziellen Firmen angestellt sind, dort aber weitgehende Freiheiten haben, dürfte die direkte Einflussnahme durch kommerzielle Interessen auf die Weiterentwicklung des Linux-Kernels noch geringer sein, als es die generierten Zahlen belegen. Trotz der fehlenden Kontrolle durch die Firmen widerlegen die dieser Arbeit zugrunde liegenden empirischen Daten das weitverbreitete und z. B. durch die MERIT-Studie (Ghosh 2006) bekräftigte Image der Sozialromantik von Open-Source-Software.

1. Schlussfolgerung

Ökonomisch relevante Open-Source-Software wie der Linux-Kernel wird zum größten Teil durch von Firmen bezahlte und bereitgestellte Entwickler vorangetrieben.

In den Interviewergebnissen hat sich gezeigt, dass das Open-Source-Entwicklungsmodell für die beteiligten Firmen mit einigen Problemen behaftet ist. Insbesondere die Koordination mit Partnern im Linuxgeschäft, die mangelnde Terminkontrolle, die komplexe Kommunikation mit der Community sowie urheberrechtliche Aspekte werden als störend bezeichnet. Anderer-

¹⁸⁴ Torvalds lässt sich ganz bewusst nicht durch eine Firma anstellen, die Linux nahesteht, wie er in seiner Biografie erklärt (Torvalds 2001). Trotzdem ist er dank Linux reich geworden, da ihm Red Hat beim Börsengang als Dank eine beträchtliche Anzahl an Aktien geschenkt hat.

seits werden als Pluspunkte die bessere Qualität der Software sowie der zur Verfügung stehende große Wissens- und Entwicklungspool außerhalb der eigenen Firma genannt. Insgesamt heben sich die positiven und negativen Aspekte des Open-Source-Entwicklungsmodells für die beteiligten Firmen weitgehend auf und würden somit das stetig zunehmende Interesse der Unternehmen an Open Source – die sich aufgrund des Lizenzmodells freier Software meist nicht mehr rückgängig machen lässt – alleine noch nicht erklären. Deshalb liegt die Annahme nahe, dass sich die Firmen aufgrund der Open Source zugrunde liegenden Idee – der Freiheit der Software – Vorteile versprechen.

2. Schlussfolgerung

Open-Source-Software wird von großen IKT-Firmen zunehmend nicht nur aus pragmatischen Motiven unterstützt, sondern aus strategischen Überlegungen aufgrund der Freiheit der Software. Der Begriff „Open“ sollte deshalb dahin gehend erweitert und verstanden werden, dass wirtschaftliches und gesellschaftliches Wachstum nicht länger durch Eigentumsschutz gewährleistet werden kann.¹⁸⁵ Gerade die Öffnung und Offenheit z. B. des Linux-Kernels erzeugt und nutzt entsprechendes Innovations- und Wachstumspotenzial.

Diese Schlussfolgerung wird, wie im Kapitel 9.1.5 erläutert, zukünftig kontinuierlich zu einer freien Software-Infrastruktur führen. Zwar werden sich die einzelnen Firmen vornehmlich in Gebieten an freien Projekten engagieren, durch die in ihrem Kerngeschäft keine direkte Konkurrenz entsteht. Wenn dies Firmen aus verschiedensten Sparten tun, wird es jedoch zu einem breiten Angebot an frei zugänglicher Software führen. IKT-Unternehmen, die sich längerfristig erfolgreich im Markt etablieren wollen, müssten demnach ihr Geschäftsmodell auf der Vermarktung des in der Diskussion erwähnten Anteils von 10 % der differenzierenden Business-Software am gesamten Software-Stack aufbauen.¹⁸⁶ Damit die Differenzierung von den

185 Materu bezeichnet in einem Report der Weltbank die aktuelle Phase als „*o-decade*“ (Materu 2004, S. 5). Er fasst mit diesem Begriff Open Source, Open Systems, Open Standards, Open Access, Open Archives und Open Everything zusammen.

186 Dass diese Erkenntnis insbesondere Microsoft außerordentlich stark im Kern ihres Geschäftsmodells trifft und sie sich in der Vergangenheit vehement gegen FLOSS gewehrt haben, ist naheliegend. Die zunehmende Öffnung gegenüber FLOSS in jüngerer Zeit ist ein deutliches Zeichen, dass auch die Firma Microsoft erkannt hat, dass sie ihr Geschäftsmodell anpassen muss, d. h., sie wendet sich zunehmend vom Vermarkten

Konkurrenten tatsächlich erreicht werden kann, muss es sich dabei zwingend um Individualsoftware handeln.

3. Schlussfolgerung

In einigen Jahren wird es eine weitgehend freie Software-Infrastruktur auch für ökonomisch relevante und geschäftskritische Bereiche geben. Differenzierende Business-Software wird als proprietäre Individualsoftware neben Freier Software koexistieren. Proprietäre Standardsoftware wird hingegen an Bedeutung verlieren.

Die beteiligten Firmen sowie die Community haben aufgrund der über die letzten rund zehn Jahre gewachsenen gemeinsamen Arbeit am Linux-Kernel ihre internen Prozesse weitgehend aufeinander abgestimmt. Zur gegenseitigen Absicherung hat sich zudem ein komplexes juristisches Netzwerk entwickelt.

4. Schlussfolgerung

Das Open-Source-Entwicklungsmodell kann beim Linux-Kernel nicht mehr als ein ausschließlich sich selbst organisierendes System verstanden werden, zumindest nicht außerhalb des engeren Entwicklungskerns. Wer das gesamte System betrachtet, sollte eher von einer „vertraglich abgesicherten Selbstorganisation“ sprechen.

Während die Sozialromantischen Hacker über ihre Identifikation mit der Ideologie von Open Source einen unerwünschten Einfluss auf Firmenentscheidungen ausüben können, besteht bei den Pragmatischen Ingenieuren die Gefahr der Professionalisierung der Community und damit der Vertreibung der Freiwilligen. Die Dialektischen Informatiker können hier eine vermittelnde Rolle einnehmen. Um ein Gleichgewicht sowohl innerhalb der Firmen als auch in der

von Infrastruktur-Software ab und dem Anbieten von differenzierenden Dienstleistungen zu. Der Verfasser wertet denn auch das Verhalten dieses Unternehmens, welches immer noch FUD („Fear, Uncertainty and Doubt“) im Zusammenhang mit Open-Source-Software zu verbreiten versucht, eher als Strategie, um Zeit zu gewinnen, damit das eigene Geschäft angepasst werden kann.

Community zu erhalten, sollten insbesondere die Firmen darauf bedacht sein, dass alle Typen in der Community gebührend vertreten sind.

5. Schlussfolgerung

Das evolutionäre Entscheidungsmodell legt nahe, dass die Strategie und das daraus abgeleitete Verhalten der Firmen einen nachhaltigen Einfluss auf die Normen und Werte der Community haben. Es hat sich gezeigt, dass Communities, die durch Firmen dominiert werden, isoliert bleiben. Daher müssen die Firmen ihre direkte und indirekte Einflussnahme sehr vorsichtig steuern, wenn sie die Bewegung nicht internalisieren wollen. Dieser Tatsache scheinen sich die großen IT-Firmen, die FLOSS strategisch unterstützen, zumindest beim Linux-Kernel bewusst zu sein.

Durch die weitere Verbreitung von Open-Source-Software nimmt auch der Bedarf an geeigneten Mitarbeitenden zu. Diese Studie hat gezeigt, dass es je nach Funktion unterschiedliche Typen von Softwareentwicklern für entsprechende betriebliche Funktionen braucht. Parallel zur Entwicklung der Diskussion zum Thema Open-Source-Software entwickeln sich auch die beruflichen Funktionen im Open-Source-Bereich. Die enthusiastischen, sozialromantischen Hacker sind zumindest beim Linux-Kernel und in den großen IT-Firmen am Verschwinden. Der größte Teil des an den Linux-Kernel übermittelten Codes wird von pragmatisch denkenden und agierenden Ingenieuren beigetragen, deren Tätigkeit sich nur marginal von der proprietären Softwareentwicklung unterscheidet. Zu ihrer Unterstützung in den Firmen bedarf es jedoch auch dialektischer Informatiker, die vermittelnd wirken und in der Open-Source-Community gut verwurzelt sind. Je nach Größe und Strategie einer Firma bedarf es mehr oder weniger beider Typen.

Open-Source-Softwareentwickler werden sowohl betriebsintern als auch -extern rekrutiert, wie die Interviewergebnisse der Linux-Kernel-Entwickler in großen IT-Firmen aufgezeigt haben. Während bei internen Wechseln durchaus Einarbeitungszeit für die Besonderheiten der Tätigkeiten im Open-Source-Bereich gewährt wird, setzt man bei externen Bewerbungen entsprechende Kenntnisse und Erfahrungen voraus. Das Wissen über die Arbeit in einer offenen Community nimmt folglich an Bedeutung stetig zu und ist deshalb auch vermehrt in der Aus-

bildung zu berücksichtigen. Aufgrund der offenen Communities bietet sich dabei eine geleitete Teilnahme an ausgewählten Open-Source-Projekten an.

6. Schlussfolgerung

Aufgrund der zunehmenden Bedeutung von Open-Source-Software wird in Zukunft priorisiert IT-Personal rekrutiert werden, das bereits über Erfahrungen in der Community-Arbeit verfügt. In der Aus- und Weiterbildung von Informatik-Mitarbeitenden nimmt die Schulung von Community-Arbeit zu. Ein ausschließlich technisch orientiertes Angebot genügt den Anforderungen zunehmend nicht mehr, vielmehr müssen auch die Soft Skills gefördert werden. Dies kann aufgrund der offen zugänglichen Communities sehr praxisnah stattfinden.

11 Ausblick

Die Aussage, dass ökonomisch relevante Open-Source-Software hauptsächlich von Firmen vorangetrieben wird (1. Schlussfolgerung), bezieht sich auf die empirischen Daten des Linux-Kernels und lässt sich nicht ohne Weiteres generalisieren. Ein möglicher Anknüpfungspunkt für künftige Studien ist folglich, diese Aussage auf eine breitere empirische Basis zu stellen. Unmittelbar damit einher geht die Anforderung, dass im Detail zu definieren ist, durch welche Eigenschaften sich „ökonomisch relevante“ FLOSS auszeichnet und welche Projekte diesen Eigenschaften entsprechen.

Die Beteiligung von Firmen an Open Source wurde bisher vorwiegend aus Sicht des Entwicklungsmodells betrachtet. Wie die Interviewergebnisse dieser Dissertation zeigen, ist jedoch die Freiheit der Software zumindest beim Linux-Kernel ebenso bedeutend für die beteiligten Firmen (2. Schlussfolgerung).¹⁸⁷ Dieser Aspekt verdient eine weitere, vertiefende Betrachtung, sowohl aus strategisch ökonomischer als auch aus juristischer Sicht.

Für die Unternehmen wird der strategische Aspekt bei der Beteiligung an FLOSS zunehmend bedeutsamer, nicht zuletzt aufgrund der prognostizierten Diffusion in der Wirtschaft (3. Schlussfolgerung). Dabei ist einerseits wichtig, sich den Community-Prozessen anzupassen. Andererseits müssen die Unternehmen darauf achten, dass sie sich ihren strategischen Spielraum weiterhin erhalten. Die vorliegende Studie zeigt auf, wie sich Firmen, die sich schon über Jahre in der Open-Source-Welt bewegen, organisiert haben (4. Schlussfolgerung). Diese Erkenntnisse könnten weiterverarbeitet und ergänzt werden, sodass auch neu einsteigende Unternehmen davon profitieren können. Dazu gehören die Optimierung der internen Organisation und Prozesse, die Etablierung von Normen und Werten, die der Community-Arbeit gerecht werden, sowie im Bereich der Human Resources die Anpassung von Rekrutierung, Weiterbildung und Teambildung.

Wie wichtig die von Firmen bezahlten Entwickler für Open-Source-Projekte sind, konnte aufgrund der empirischen Daten aufgezeigt werden. Die Linux-Kernel-Community hat es dabei vorzüglich geschafft, ihre Prozesse an diese Gegebenheit anzupassen, ohne dass die Philoso-

¹⁸⁷ Ein sehr aktuelles Beispiel für diese These ist das für Netbooks gedachte Betriebssystem ChromeOS, das zwar Open Source ist, aber weitgehend von Google intern entwickelt wird. Siehe dazu u.a. den Blogeintrag unter <http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html> [27.08.2009].

phie der Freien Software darunter gelitten hat. Obwohl jedes Open-Source-Projekt seinen eigenen Charakter hat, ließe sich aus diesen Erkenntnissen ein Leitfaden für die Zusammenarbeit der Community mit Firmen erstellen.

Die drei Typen von Open-Source-Softwareentwicklern im unternehmerischen Umfeld – der Pragmatische Ingenieur, der Dialektische Informatiker und der Sozialromantische Hacker – wurden aufgrund eines qualitativen Vorgehens eruiert. Aussagen über das quantitative Auftreten der Typen sind deshalb nicht möglich. Eine ergänzende, quantitative Studie – die zudem über den Linux-Kernel hinausgeht – wäre deshalb wünschenswert. Das Wissen über die quantitative Verteilung der Typen ist insofern von Bedeutung, als ein ausgewogenes Verhältnis zu einer Stabilisierung der Community beiträgt (5. Schlussfolgerung).

Für die Ausbildung insbesondere an Hochschulen, aber auch für die berufliche Weiterbildung sollte die Arbeit in der Open-Source-Community ein größeres Gewicht erhalten (6. Schlussfolgerung). Ein erster Schritt ist sicherlich die an der Universität Erlangen-Nürnberg kürzlich ausgeschriebene Professur für Open-Source-Software oder der in Lissabon gestartete Masterstudiengang für Open-Source-Software. Da FLOSS jedoch immer flächendeckender eingesetzt wird, sollte das Thema auch in die allgemeine Informatikausbildung integriert werden. Diesbezüglich wären entsprechende Kompetenzen festzulegen und Lernziele sowie Lehrpläne zu erstellen.

12 Literaturverzeichnis

- Altheide, D.L. und Johnson, J.M. (1994). Criteria for Assessing Interpretive Validity in Qualitative Research. In: N. K. Denzin und Y. S. Lincoln (Hrsg.), *Handbook of Qualitative Research*. Thousand Oaks : Sarge, S. 485-499.
- Anderson, C. (2006). *The Long Tail Why the Future of Business Is Selling Less of More*. New York : Hyperion.
- Auener, D. (2008). *Vertikale Integration von IT-Dienstleistungen durch den Einsatz offener Software : Am Fallbeispiel des Auswärtigen Amts*. Diplomarbeit. TU Berlin.
- Baldwin, C.Y. und Clark, K.B. (2006). The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? *Manage. Sci.*, 52(7), 1116-1127.
- Bauer, A. und Pizka, M. (2005). Der Beitrag freier Software zur Software-Evolution. In: B. Lutterbeck, R. A. Gehring und M. Bärwolff (Hrsg.), *Open Source Jahrbuch 2005 : Zwischen Softwareentwicklung und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 95-112.
- Baumgärtel, T. (2002). Am Anfang war alle Software frei. Microsoft, Linux und die Rache der Hacker. In: A. Roesler und B. Stiegler (Hrsg.), *Microsoft : Medien, Macht, Monopol*. Frankfurt/Main : Suhrkamp, S. 103-129.
- Becker, G.S. (1993). *Der Ökonomische Ansatz Zur Erklärung Menschlichen Verhaltens*. Tübingen : Mohr.
- Beck, K. (2000). *Extreme Programming : Das Manifest*. München : Addison-Wesley.
- Bell, D. (1976). *Die nachindustrielle Gesellschaft*. Frankfurt : Campus.
- Benabou, R. und Tirole, J. (2000). Self-Confidence and Social Interactions. *SSRN eLibrary*.
- Benkler, Y. (2003). Coase's Penguin, or, Linux and The Nature of the Firm. *The Yale Law Journal*, 112, 369-446.
- Bernard, R.H. (2000). *Social Research Methods : Qualitative and Quantitative Approaches*. Thousand Oaks : Sage.
- Bezroukov, N. (1999a). Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism). *First Monday*, 4(10). Online: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/696/606> [09.09.2009].
- Bezroukov, N. (1999b). A Second Look at the Cathedral and the Bazaar. *First Monday*, 4(12). Online: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/708/618> [09.09.2009].
- Bhagwati, J.N. (2008). *Verteidigung der Globalisierung*. München : Pantheon Verlag.

- BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (2007a). *Standortnachteil Fachkräftemangel: Fakten und Lösungsansätze : Wie Politik, Wirtschaft und Wissenschaft den Hightech-Standort Deutschland nachhaltig stärken könnten*. Berlin : BITKOM.
- BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (2007b). *Standpunkte zur Zuwanderung hochqualifizierter Arbeitskräfte : Den Wettbewerb um die besten Köpfe gewinnen*. Berlin : BITKOM.
- BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (2009). *Wachstumskräfte stärken. Die Hightech-Agenda für die 17. Wahlperiode*. Berlin : BITKOM.
- Boehm, B.W. (2000). *Software Cost Estimation with Cocomo II*. Upper Saddle River, N.J : Prentice Hall.
- Bogner, A. und Menz, W. (2002a). Das theoriegenerierende Experteninterview : Erkenntnisinteresse, Wissensformen, Interaktion. In: A. Bogner und W. Menz (Hrsg.), *Das Experteninterview : Theorie, Methode, Anwendung*. Opladen : Leske + Budrich, S. 33-70.
- Bogner, A. und Menz, W. (2002b). Expertenwissen und Forschungspraxis: die modernisierungstheoretische und die methodische Debatte um die Experten. Zur Einführung in ein unübersichtliches Problemfeld. In: A. Bogner und W. Menz (Hrsg.), *Das Experteninterview : Theorie, Methode, Anwendung*. Opladen : Leske + Budrich, S. 7-29.
- Bohnsack, R. (1992). Dokumentarische Interpretation von Orientierungsmustern : Verstehen-Interpretieren-Typenbildung in wissenssoziologischer Analyse. In: M. Meuser und R. Sackmann (Hrsg.), *Analyse sozialer Deutungsmuster : Beiträge zur empirischen Wissenssoziologie*. Pfaffenweiler : Centaurus, S. 139-160.
- Bonaccorsi, A. und Rossi, C. (2003). Why Open Source Software Can Succeed. *Research Policy*, 32, 1243–1258.
- Bourdieu, P. (Hrsg.). (1997). *Das Elend der Welt : Zeugnisse und Diagnosen alltäglichen Leidens an der Gesellschaft*. Konstanz : UVK Universitätsverlag Konstanz.
- Brand, S. (1988). *The Media Lab: Inventing the Future at M. I. T.* New York : Penguin Books.
- Brecht, B. (1967). Rede über die Funktion des Rundfunks. In: *Gesammelte Werke - Band VIII*. Frankfurt am Main : Suhrkamp, S. 127-134.
- Brooks, F.P.J. (1995). *The Mythical Man-Month*. Boston : Addison-Wesley.
- Carbone, P. (2006). Competing with Open Source Software : Insights from Recent Research. In: *OCRI Partnership Conference Series*. Ottawa. Online: <http://www.ocri.ca/events/presentations/partnership/April2106/Carbone.pdf> [20.08.2009].
- Carbone, P. (2007). Value Derived from Open Source is a Function of Maturity Levels. In:

- OCRI Partnership Conference Series*. Ottawa. Online: <http://www.ocri.ca/events/presentations/partnership/April1907/PeterCarbone.pdf> [20.08.2009].
- Castells, M. (2001). *Das Informationszeitalter : Der Aufstieg der Netzwerkgesellschaft*. Opladen : Leske + Budrich.
- Chesbrough, H.W. (2003). *Open Innovation the New Imperative for Creating and Profiting from Technology*. Boston : Harvard Business School Press.
- Coase, R.H. (1937). The Nature of the Firm. *Economica*, 4(16), 386-405.
- Cohen, S. (2008). Open Source: The Model Is Broken. *BusinessWeek*. Online: http://www.businessweek.com/technology/content/nov2008/tc20081130_276152.htm [20.08.2009].
- Dahlander, L. und Magnusson, M.G. (2006). Business Models and Community Relationships of Open Source Software Firms. In: J. Bitzer und P. J. H. Schröder (Hrsg.), *The Economics of Open Source Software Development*. Amsterdam : Elsevier, S. 111-130.
- Dahlander, L. und Wallin, M.W. (2006). A man on the inside: Unlocking communities as complementary assets. *Research Policy*, 35, 1243-1259.
- Deci, E.L. und Ryan, R.M. (1987). The support of autonomy and the control of behavior. *Journal of Personality and Social Psychology*, 53(6), 1024-1037.
- DeMarco, T. und Lister, T. (1999). *Wien wartet auf Dich! : Der Faktor Mensch im DV-Management*. München : Hanser.
- Dempsey, B.J., Weiss, D., Jones, P. und Greenberg J. (1999). A Quantitative Profile of a Community of Open Source Linux Developers. *School of Information and Library Service*, TR-1999-065. Online: <http://www.ibiblio.org/osrt/develop.html> [20.08.2009].
- Denzin, N.K. (1989). *The Research Act a Theoretical Introduction to Sociological Methods*. Englewood Cliffs, N.J : Prentice-Hall.
- Diedrich, O. (2009). *Trendstudie Open Source : Wie Open-Source-Software in Deutschland eingesetzt wird*. Heise Verlag. Online: <http://www.heise.de/open/artikel/126682> [20.08.2009].
- Drucker, P.F. (1998). Management's New Paradigms. *Forbes*, (10.05.1998), 152-177.
- Drucker, P.F. (1993). *Die Postkapitalistische Gesellschaft*. Düsseldorf : Econ Verlag.
- Drucker, P.L. (1996). Licensing Alternatives for Freely Redistributable Software. In: *Freely Redistributable Software Conference*. Boston. Online: <http://gd.tuwien.ac.at/publishing/ghostscript/papers/frs96.ps> [27.05.2009].
- Eisenhardt, K.M. (1989). Building Theories from Case Study Research. *Academy of Management Review*, 14(4), 532-550.
- Elster, J. (1986). *Rational Choice*. Oxford : Basil Blackwell.

- Esser, H. (2000). *Soziales Handeln*. Frankfurt am Main : Campus Verlag.
- Ettrich, M. (2004). Koordination und Kommunikation in Open-Source-Projekten. In: R. A. Gehring und B. Lutterbeck (Hrsg.), *Open Source Jahrbuch 2004 : Zwischen Softwareentwicklung und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 179-192.
- Evans, P. und Wolf, B. (2005). Vertrauen ist die Basis. *Harvard Business Manager*, 11, 61-74.
- Evers, S. (2008). *Ein Modell der Open-Source-Entwicklung*. Dissertation. TU Berlin.
- Fehr, E. und Gächter, S. (2002). Do Incentive Contracts Undermine Voluntary Cooperation? *SSRN eLibrary*. Online: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=313028 [09.09.2009].
- Feller, J. und Fitzgerald, B. (2002). *Understanding Open Source Software Development*. London : Addison-Wesley.
- Fielding, R.T. (1999). Shared leadership in the Apache project. *Commun. ACM*, 42(4), 42-43.
- Fielding, R.T. (2008). Open Architecture. In: *O'Reilly OSCON Open Source Convention*. Online: <http://assets.en.oreilly.com/1/event/12/Open%20Architecture%20at%20REST%20Presentation.pdf> [20.08.2009].
- Flick, U. (2002). *Qualitative Sozialforschung : Eine Einführung*. Reinbek bei Hamburg : Rowohlt.
- Foddy, W. (2003). *Constructing Questions for Interviews and Questionnaires : Theory and Practice in Social Research*. Cambridge : Cambridge University Press.
- Fogel, K. (2006). *Producing Open Source Software*. Sebastopol : O'Reilly.
- Fontana, A. und Frey, J.H. (1994). Interviewing: The Art of Science. In: N. K. Denzin und Y. S. Lincoln (Hrsg.), *Handbook of Qualitative Research*. Thousand Oaks : Sage, S. 361-376.
- Fornefeld, M. und Gasper, M. (2009). *Potenzialanalyse im Technologiefeld Open Source in der Hauptstadtregion Berlin*. Berlin : TSB Innovationsagentur Berlin GmbH. Online: <http://www.tsb-berlin.de/data/files/Downloads/Studien-Potenzialanalyse.pdf> [20.08.2009].
- Forrester Consulting (Hrsg.) (2008). *Open Source Paves The Way For The Next Generation Of Enterprise IT*. Cambridge : Forrester Consulting.
- Frey, B.S. und Goette, L. (1999). Does Pay Motivate Volunteers? *Working Papers Series*, 22. Online: <http://www.iew.unizh.ch/wp/iewwp007.pdf> [20.08.2009].
- Frey, B.S. und Stutzer, A. (2007). *Economics and Psychology a Promising New Cross-Disciplinary Field*. Cambridge, Mass : MIT Press.
- Froschauer, U. und Lueger, M. (2002). ExpertInnengespräche in der interpretativen Organisationsforschung. In: A. Bogner und W. Menz (Hrsg.), *Das Experteninterview :*

- Theorie, Methode, Anwendung*. Opladen : Leske + Budrich, S. 223-240.
- Gerhardt, U. (2001). *Idealtypus : Zur methodischen Begründung der modernen Soziologie*. Frankfurt am Main : Suhrkamp.
- Ghosh, R.A. (1998). Cooking pot markets: an economic model for the trade in free goods and services on the Internet. *First Monday*, 3(3). Online: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/580/501> [09.09.2009].
- Ghosh, R.A. (2006). *Study on the: Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU*. MERIT. Online: <http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf> [20.08.2009].
- Ghosh, R.A., Glott, R. und Robles, G. (2002). *The Free/Libre and F/OSS Software Developers Survey and Study - FLOSS Final Report*. Maastricht : International Institute of Infonomics, University of Maastricht and Berlecon Research GmbH. Online: <http://www.infonomics.nl/FLOSS/report/> [20.08.2009].
- Ghosh, R.A. und Prakash, V.V. (2000). The Orbiten free software survey. *First Monday*, 5(7). Online: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/769> [09.09.2009].
- Gibson, W. (2000). *Die Neuromancer-Trilogie. Drei Romane: Neuromancer / Biochips / Mona Lisa Overdrive*. München : Heyne.
- Glaser, B.G. und Strauss, A.L. (1967). *The Discovery of Grounded Theory Strategies for Qualitative Research*. New York : Aldine.
- Godfrey, M.W. und Tu, Q. (2000). Evolution in Open Source Software: A Case Study. In: *In Proceedings of the International Conference on Software Maintenance*. S. 131-142.
- Grassmuck, V. (2002). *Freie Software Zwischen Privat- und Gemeineigentum*. Bonn : Bundeszentrale für politische Bildung.
- Hardin, G. (1968). The Tragedy of the Commons. *Science*, 162(3859), 1243-1248.
- Hars, A. und Ou, S. (2002). Working for Free? - Motivations of Participating in Open Source Projects. *International Journal of Electronic Commerce*, 6, 25-39.
- Healy, K. und Schussman, A. (2003). The ecology of open-source software development. *Department of Sociology, University of Arizona*, 23. Online: <http://opensource.mit.edu/papers/healyschussman.pdf> [16.06.2009].
- Helfferrich, C. (2005). *Die Qualität qualitativer Daten : Manual für die Durchführung qualitativer Daten*. Wiesbaden : VS Verlag für Sozialwissenschaften.
- Henkel, J. (2009). Champions of Revealing - the Role of Open Source Developers in Commercial Firms. *Industrial and Corporate Change*, 18(3), S. 435-471.
- Hertel, G., Niedner, S. und Herrmann, S. (2003). Motivation of software developers in Open

Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7), 1159-1177.

Herzberg, F. (1959). *The Motivation to Work*. New York : John Wiley.

Himanen, P. (2001). *The Hacker Ethic and the Spirit of the Information Age*. London : Secker & Warburg.

von Hippel, E. und von Krogh, G. (2003). Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*, 14(2), 209-223.

Hopf, C. (2000). Qualitative Interviews - ein Überblick. In: U. Flick, E. von Kardoff und I. Steinke (Hrsg.), *Qualitative Forschung : Ein Handbuch*. Reinbek bei Hamburg : Rowohlt, S. 349-360.

Hunt, A. und Thomas, D. (2003). *Der pragmatische Programmierer*. München : Hanser.

Imhorst, C. (2004). *Die Anarchie der Hacker : Richard Stallman und die Freie-Software-Bewegung*. Marburg : Tectum.

Janoska-Bendl, J. (1965). *Methodologische Aspekte des Idealtypus : Max Weber und die Soziologie der Geschichte*. Berlin : Duncker & Humblot.

Jollans, A. (2006). Open Source Beyond Linux : Collaborative Innovation for your business. *Linux@IBM Event 27.10.2006, Zürich*.

Kaesler, D. (2003). *Max Weber : Eine Einführung in Leben, Werk und Wirkung*. Frankfurt am Main : Campus.

Kelle, U. und Kluge, S. (1999). *Vom Einzelfall zum Typus : Fallvergleich und Fallkontrastierung in der qualitativen Sozialforschung*. Opladen : Leske + Budrich.

Klein, N. (2001). *No Logo! Der Kampf der Global Players um Marktmacht: Ein Spiel mit vielen Verlierern und wenigen Gewinnern*. München : Riemann.

Kollock, P. (1999). The economics of on line cooperation: gift and public goods in cyberspace. In: M. A. Smith und P. Kollock (Hrsg.), *Communities in cyberspace*. London : Routledge, S. 220-246.

Kowal, S. und O'Connell, D.C. (2000). Zur Transkription von Gesprächen. In: U. Flick, E. von Kardoff und I. Steinke (Hrsg.), *Qualitative Forschung : Ein Handbuch*. Reinbek bei Hamburg : Rowohlt, S. 437-447.

Kreft, U. (2008). Burnout in der IT-Branche. *ITG-Arbeitspapier*, 02. Online: http://www.risp-duisburg.de/abtpro/prolog/ap2_itg_final.pdf [20.08.2009].

Krishnamurthy, S. (2002). Cave or community? *First Monday*, 7(6). Online: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/960/881> [09.09.2009].

Kroah-Hartman, G. (2007). Linux Kernel Development : How Fast it is Going, Who is Doing

- It, What They are Doing, and Who is Sponsoring It. In: *Proceedings of the Linux Symposium : Volume One*. S. 239-244. Online: <https://ols2006.108.redhat.com/2007/Reprints/kroah-hartman-Reprint.pdf> [20.08.2009].
- Kroah-Hartman, G., Corbet, J. und McPherson, A. (2008). *Linux Kernel Development : How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*. Online: <http://www.linuxfoundation.org/publications/linuxkerneldevelopment.pdf> [07.06.2009].
- Kroah-Hartman, G., Corbet, J. und McPherson, A. (2009). *Linux Kernel Development : How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It: An August 2009 Update*. Online: <http://www.linuxfoundation.org/publications/whowriteslinux.pdf> [19.08.2009].
- Kruse, J. (2006). *Reader "Qualitative Interviewforschung"*. Freiburg.
- Kuwabara, K. (2000). Linux: A bazaar at the edge of chaos. *First Monday*, 5(3). Online: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/1482/1397> [09.09.2009].
- Kvale, S. (1996). *InterViews : An introduction to qualitative research interviewing*. Thousand Oaks : Sage Publications.
- Laisné, J. (Hrsg.) (2008). *2020 FLOSS Roadmap*. Paris. Online: http://www.2020flossroadmap.org/docs/OWF_2020_Roadmap%20v2.18-3.pdf [09.09.2009].
- Lakhani, K. und von Hippel, E. (2003). How Open Source software works: "Free" user-to-user assistance. *Research Policy*, 32, 923-943.
- Lakhani, K.R. und Wolf, R.G. (2003). *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects*. Boston : MIT Sloan.
- Lammers, S. (1986). *Programmers at Work*. Microsoft Press,U.S.
- Lancashire, D. (2001). The Fading Altruism of Open Source Development. *First Monday*, 6(12).
- Lee, G.K. und Cole, R.E. (2003). From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development. *Organization Science*, 14(6), 633-649.
- Lehman, M.M. e (1997). Metrics and Laws of Software Evolution : The Nineties View. In: *Software Metrics Symposium, 1997. Proceedings., Fourth International*. Albuquerque, NM, S. 20-32.
- Leiteritz, R. (2004). Open Source-Geschäftsmodelle. In: B. Lutterbeck und R. A. Gehring (Hrsg.), *Open Source Jahrbuch 2004 : Zwischen Softwareentwicklung und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 139-170.
- Lerner, J. und Tirole, J. (2001). The open source movement: Key research questions. *European Economic Review*, 45(4-6), 819-826.

- Lerner, J. und Tirole, J. (2002). Some Simple Economics of Open Source. *Journal of Industrial Economics*, L(2), 197-234.
- Levy, S. (1994). *Hackers: Heroes of the Computer Revolution*. London : Penguin.
- Lincoln, Y.S. und Guba, E.G. (1985). *Naturalistic Inquiry*. Newbury Park : Sage.
- Luthiger Stoll, B. (2006). *Spas und Software-Entwicklung zur Motivation von Open-Source-Programmierern*. Dissertation. Zürich.
- Lutterbeck, B. (2005). Infrastrukturen der Allmende - Open Source, Innovation und die Zukunft des Internets. In: B. Lutterbeck, R. A. Gehring und M. Bärwolff (Hrsg.), *Open Source Jahrbuch 2005 : Zwischen Softwareentwicklung und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 329-346.
- Lutterbeck, B. (2006). Die Zukunft der Wissensgesellschaft. In: J. Hofmann (Hrsg.), *Wissen und Eigentum : Geschichte, Recht und Ökonomie stoffloser Güter*. Bonn : bpb Bundeszentrale für politische Bildung, S. 319-340.
- Lutterbeck, B. (2007). *IT-Strategie des Landes Berlin: Open Source/Offene Standards beim Einsatz von Software in der Öffentlichen Verwaltung : Anhörung des Ausschusses für Verwaltungsreform, Kommunikations- und Informationstechnik des Abgeordnetenhauses Berlin, 3. Mai 2007*. Berlin : TU Berlin. Online: <http://ig.cs.tu-berlin.de/ma/bl/-OffeneStandardsOffeneSoftwareWarumDerSenatVonBerlinSichFuerLinuxEntscheidenSollte-2007-05-03.pdf> [22.02.2009]
- Lutterbeck, B., Bärwolff, M. und Zimmermann, B. (2007). *Open Source und offene Standards beim Einsatz von Software in der öffentlichen Verwaltung : Expertise im Rahmen der Anhörung des Abgeordnetenhauses von Berlin am 3. Mai 2007*. Berlin. Online: <http://ig.cs.tu-berlin.de/ma/bl/ap/2007/-OpenSourceUndOffeneStandardsBeimEinsatzVonSoftwareInDeroeffentlichenVerwaltung-2007-05-03.pdf> [22.02.2009].
- Madsen, M.R. (2009). *Open Source in the Business Intelligence Market : Crossing the Threshold to Mainstream Adoption*. Rogue River : Third Nature.
- Maillart, T., Sornette, D., Spaeth, S. und von Krogh, G. (2008). Empirical Tests of Zipf's law Mechanism In Open Source Linux Distribution. *Physical Review Letters*, 101 (218701).
- Malone, T.W. (2004). *The Future of Work How the New Order of Business Will Shape Your Organization, Your Management Style, and Your Life*. Boston : Harvard Business School Press.
- Markus, M.L., Manville, B. und Agres, C.E. (2000). What makes a virtual organization work? *MIT Sloan Management Review*, 42(1), 13-26.
- Materu, P.N. (2004). *Open Source Courseware : A Baseline Study*. Washington, DC : Weltbank. Online: http://siteresources.worldbank.org/INTAFRREGTOPTIEIA/Resources/open_source_courseware.pdf [09.09.2009].
- Mayring, P. (1997). *Qualitative Inhaltsanalyse*. Weinheim : Deutscher Studien Verlag.

- Merkens, H. (2000). Auswahlverfahren, Sampling, Fallkonstruktion. In: U. Flick, E. von Kardoff und I. Steinke (Hrsg.), *Qualitative Forschung : Ein Handbuch*. Reinbek bei Hamburg : Rowohlt, S. 286-299.
- Merton, R.K., Fiske, M. und Kendall, P.L. (1990). *The Focused Interview a Manual of Problems and Procedures*. New York : The Free Press.
- Metcalf, B. (1999). Linux's '60s technology, open-sores ideology won't beat W2K, but what will? *InfoWorld*, 21(25), S. 67-69.
- Meuser, M. und Nagel, U. (2002a). Vom Nutzen der Expertise : ExpertInneninterviews in der Sozialberichterstattung. In: A. Bogner und W. Menz (Hrsg.), *Das Experteninterview : Theorie, Methode, Anwendung*. Opladen : Leske + Budrich, S. 257-272.
- Meuser, M. und Nagel, U. (2002b). ExpertInneninterviews - vielfach erprobt, wenig bedacht : Ein Beitrag zur qualitativen Methodendiskussion. In: A. Bogner und W. Menz (Hrsg.), *Das Experteninterview : Theorie, Methode, Anwendung*. Opladen : Leske + Budrich, S. 71-93.
- Mockus, A., Fielding, R.T. und Herbsleb, J. (2000). A case study of open source software development: the Apache server. In: *Proceedings of the 22nd international conference on Software engineering*. Limerick, Ireland : ACM, S. 263-272.
- Moody, G. (2001). *Die Software-Rebellen : Die Erfolgsstory von Linus Torvalds und Linux*. Landsberg/Lech : Verlag Moderne Industrie.
- Moody, G. (2008). Richards Stallmans Goldene Regel und das "Digital Commons". In: B. Lutterbeck, M. Bärwolff und R. A. Gehring (Hrsg.), *Open Source Jahrbuch 2008 : Zwischen freier Software und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 299-308.
- Moon, J.Y. und Sproull, L. (2002). Essence of Distributed Work: The Case of the Linux Kernel. In: P. Hind und S. Kiesler (Hrsg.), *Distributed Work*. Cambridge, Mass. : MIT Press, S. 381-404.
- Nafus, D., Leach, J. und Krieger, B. (2006). *FLOSSPOLs Deliverable D 16 - Gender: Integrated Report of Findings*. Cambridge : UCAM, University of Cambridge. Online: http://flosspols.org/deliverables/FLOSSPOLs-D16-Gender_Integrated_Report_of_Findings.pdf [20.08.2009].
- Nakakoji, K. e (2002). Evolution patterns of open-source software systems and communities. In: *IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution*. ACM Press, S. 85, 76.
- Neumann, P. (2000). Robust Nonproprietary Software. In: *IEEE Symposium on Security and Privacy*. Oakland, CA.
- North, D.C. (1990). *Institutions, institutional change, and economic performance*. Cambridge : University Press.
- North, D.C. (1992). *Transaction Costs, Institutions, and Economic Performance*. San

Francisco : International Center for Economic Growth.

- Ohno, T. (1993). *Das Toyota-Produktionssystem*. Frankfurt/Main : Campus.
- O'Mahony, S. (2002). *Community Managed Software Projects : The Emergence of a New Commercial Actor*. Dissertation. Stanford University.
- O'Mahony, S. (2003). Guarding the commons: how community managed software projects protect their work. *Research Policy*, 32(7), 1179-1198.
- O'Mahony, S., Fernando, C.D. und Mamas, E. (2005). *IBM and Eclipse*. Harvard : Harvard Business School.
- Oreizy, P. (2000). *Open Architecture Software: A Flexible Approach to Decentralized Software Evolution*. Dissertation. University of California, Irvine. Online: <http://www.ics.uci.edu/%7Epeyman/papers/thesis.pdf.gz> [09.09.2009].
- Osterloh, M. und Weibel, A. (2006). *Investition Vertrauen : Prozesse der Vertrauensentwicklung*. Wiesbaden : Gabler.
- Perens, B. (2007). Open Source ein aufstrebendes ökonomisches Modell. In: B. Lutterbeck, M. Bärwolf und R. A. Gehring (Hrsg.), *Open Source Jahrbuch 2007: Zwischen Softwareentwicklung und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 131-164.
- Platt, J. (2001). The History of the Interview. In: J. F. Gubrium und J. A. Holstein (Hrsg.), *Handbook of Interview Research : Context & Method*. Thousand Oaks : Sage, S. 33-54.
- Popper, K.R. (1993). Was ist Dialektik? In: E. Topitsch (Hrsg.), *Logik der Sozialwissenschaften*. Königstein : Athenäum - Hain - Hanstein, S. 262-290.
- Probst, G.J.B. (1987). *Selbst-Organisation Ordnungsprozesse in Sozialen Systemen Aus Ganzheitlicher Sicht*. Berlin : P. Parey.
- Rähm, J. (2009). Die Konferenz Berlin Open soll die Idee der freien Software neu beleben. Online: http://ondemand-mp3.dradio.de/file/dradio/2009/06/20/dlf_20090620_1635_be8409db.mp3 [07.06.2009].
- Raymond, E.S. (1999). *The Cathedral & the Bazaar : Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol : O'Reilly.
- Renner, T., Vetter, M., Rex, S. und Kett, H. (2005). *Open Source Software : Einsatzpotentiale und Wirtschaftlichkeit*. Stuttgart : Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO.
- Richter, M. (2006). Fair Code - Freie/Open-Source-Software und der Digital Divide? In: B. Lutterbeck, M. Bärwolf und R. A. Gehring (Hrsg.), *Open Source Jahrbuch 2006 : Zwischen Softwareentwicklung und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 371-380.
- Rifkin, J. (2000). *Access : Das Verschwinden des Eigentums*. Frankfurt/Main : Campus.

- Roberts, J., Hann, I. und Slaughter, S. (2006). Understanding the Motivations, Participation and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Marshall School of Business Working Paper No. IOM 01-06*. Online: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=918518 [09.09.2009].
- Rosenberg, D.K. (2000). *Open Source: The Unauthorized White Papers*. Foster City, CA : M&T Books.
- Rossi, C. und Bonaccorsi, A. (2006). Intrinsic Motivations and Profit-Oriented Firms in Open Source Software: Do Firms Practise What they Preach? In: J. Bitzer und P. J. H. Schröder (Hrsg.), *The Economics of Open Source Software Development*. Amsterdam : Elsevier, S. 83-109.
- Rossi, M.A. (2006). Decoding the Free/Open Source Software Puzzle: A Survey of Theoretical and Ampirical Contributions. In: J. Bitzer und P. J. H. Schröder (Hrsg.), *The Economics of Open Source Software Development*. Amsterdam : Elsevier, S. 15-55.
- Schumpeter, J.A. (1961). *Konjunkturzyklen: Eine Theoretische, historische und statistische Analyse des Kapitalistischen Prozesses*. Göttingen : Vandenhoeck und Ruprecht.
- Smith, A. (1974). *Der Wohlstand der Nationen : Eine Untersuchung seiner Natur und seiner Ursachen*. München : Beck.
- Smith, J.M. (1982). *Evolution and the Theory of Games*. Cambridge : Cambridge University Press.
- Spaeth, S., Stuermer, M. und von Krogh, G. (2008). Enabling Knowledge Creation through Outsiders: Towards a Push Model of Open Innovation. In: *International Journal of Technology Management (forthcoming)*.
- Spinellis, D. (2008). A Tale of Four Kernels. In: *Proceedings of the 30th international conference on Software engineering*. Leipzig, Germany : ACM, S. 381-390.
- Stake, R.E. (1994). Case Studies. In: N. K. Denzin und Y. S. Lincoln (Hrsg.), *Handbook of Qualitative Research*. Thousand Oaks : Sarge, S. 236-247.
- Stallman, R.M. (2002). *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Boston : Free Software Foundation.
- Steinke, I., Flick, U. und von Kardoff, E. (2000). Gütekriterien qualitativer Forschung. In: *Qualitative Forschung : Ein Handbuch*. Reinbek bei Hamburg : Rowohlt, S. 319-331.
- Stephenson, N. (1995). *Snow Crash*. München : Goldmann.
- Strauss, A. und Corbin, J. (1996). *Grounded Theory: Grundlagen Qualitativer Sozialforschung*. Weinheim : Psychologie Verlags Union.
- Suhr, J. (2008). Messung von Offenheit an IT-Artefakten Der Information Technology Openness Benchmark. In: B. Lutterbeck, M. Bärwolff und R. A. Gehring (Hrsg.), *Open Source Jahrbuch 2008 : Zwischen freier Software und Gesellschaftsmodell*.

- Berlin : Lehmanns Media, S. 169-183.
- Torvalds, L. (1999). The Linux Edge. In: C. DiBona, S. Ockman und M. Stone (Hrsg.), *Open Sources: Voices from the Revolution*. Sebastopol : O'Reilly, S. 101-111.
- Torvalds, L. (2001). *Just for Fun the Story of an Accidental Revolutionary*. New York : Texere.
- Trinczek, R. (2002). Wie befrage ich Manager? : Methodische und methodologische Aspekte des Experteninterviews als qualitativer Methode empirischer Sozialforschung. In: A. Bogner und W. Menz (Hrsg.), *Das Experteninterview : Theorie, Methode, Anwendung*. Opladen : Leske + Budrich, S. 209-222.
- Valimaki, M. (2003). Dual Licensing in Open Source Software Industry. *Systemes d'Information et Management*, 8 (1), S. 63-75. Online: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1261644 [09.09.2009].
- Varian, H. und Shapiro, C. (2007). The Economics of Software Markets. In: B. Lutterbeck, M. Bärwolff und R. A. Gehring (Hrsg.), *Open Source Jahrbuch 2007 : Zwischen freier Software und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 125-130.
- Vile, D. und Atherton, M. (2009). *Linux on the Desktop : Lessons from mainstream business adoption*. Hampshire : Freeform Dynamics Ltd. Online: http://www.freeformdynamics.com/fullarticle_subscribe.asp?aid=678 [25.08.2009].
- Weber, K. (2004). Philosophische Grundlagen und mögliche Entwicklungen der Open-Source- und Free-Software-Bewegung. In: R. A. Gehring und B. Lutterbeck (Hrsg.), *Open Source Jahrbuch 2004 : Zwischen Softwareentwicklung und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 369-383.
- Weber, M. (1934). *Die protestantische Ethik und der Geist des Kapitalismus - Die protestantischen Sekten und der Geist des Kapitalismus - Die Wirtschaftsethik der Weltreligionen*. Tübingen : Mohr.
- Weber, M. (1951). *Gesammelte Aufsätze zur Wissenschaftslehre*. Tübingen : J.C.B. Mohr.
- Weber, S. (2000). The Political Economy of Open Source Software. *BRIE Working Paper 140*. Online: <http://brie.berkeley.edu/publications/wp140.pdf> [09.09.2009].
- Weibel, A., Rost, K. und Osterloh, M. (2007). Crowding-out of intrinsic motivation - Opening the black box. *Working Paper*. Online: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=957770 [09.09.2009].
- Wenger, E. (2002). *Cultivating Communities of Practice a Guide to Managing Knowledge*. Boston, Mass : Harvard Business School Press.
- Werner, T. (2007). World Domination: Die Erfolgsgeschichte der Linux- und Open-Source-Einführung im Auswärtigen Amt. In: B. Lutterbeck, M. Bärwolff und R. A. Gehring (Hrsg.), *Open Source Jahrbuch 2007 : Zwischen freier Software und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 239-248.

- West, J. (2003). How open is open enough? : Melding proprietary and open source platform strategies. *Research Policy*, 32(7), 1259-1285.
- West, J. (2008). Unternehmen zwischen Offenheit und Profitstreben. In: B. Lutterbeck, M. Bärwolff und R. A. Gehring (Hrsg.), *Open Source Jahrbuch 2008 : Zwischen freier Software und Gesellschaftsmodell*. Berlin : Lehmanns Media, S. 83-96.
- West, J. und Gallagher, S. (2006). Patterns of Open Innovation in Open Source Software. In: H. Chesbrough, W. Vanhaverbeke und J. West (Hrsg.), *Open Innovation: Reseraching a New Paradigm*. New York : Oxford University Press, S. 82-106.
- West, J. und O'Mahony, S. (2005). Contrasting Community Building in Sponsored and Community Founded Open Source Projects. Online: <http://opensource.mit.edu/papers/westomahony.pdf> [17.08.2009].
- Wheeler, D.A. (2001). More Than a Gigabuck: Estimating GNU/Linux's Size. Online: <http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html> [17.08.2009].
- Wichmann, T. (2002). Firms' Open Source Activities: Motivations and Policy Implications. In: *FLOSS Final Report : Free/Libre Open Source Software: Survey and Study*. Berlin : Berlecon Research, S. 1-34. Online: http://www.berlecon.de/studien/downloads/200207FLOSS_Activities.pdf [17.08.2009].
- Witzel, A. (2000). Das problemzentrierte Interview. *Forum Qualitative Sozialforschung*, 1(1), 1-9. Online: <http://www.qualitative-research.net/index.php/fqs/article/view/1132/2519> [17.08.2009].
- Wolf, U. (2002). Häuptlinge und Indianer. *Linux-Magazin*, 10, 48-49.
- Xu, J., Gao, Y., Christley, S. und Madey, G. (2005). A Topological Analysis of the Open Souce Software Development Community. In: *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 07*. IEEE Computer Society, S. 198.1. Online: <http://portal.acm.org/citation.cfm?id=1043101> [18.06.2009].
- Yin, R.K. (2003). *Case study research : design and methods*. Thousand Oaks : Sage.
- Young, R. (1999). Giving It Away : How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry. In: C. DiBona, S. Ockman und M. Stone (Hrsg.), *Open Sources: Voices from the Revolution*. Sebastopol : O'Reilly, S. 113-125.

Anhang

A *Brief an Firmen*

Prof. Dr. iur. Bernd Lutterbeck

Technische Universität Berlin

Institut für Wirtschaftsinformatik



Donnerstag, 7. Juni 2007

Fallstudie Open Source und grosse ICT-Unternehmen

Sehr geehrter Herr 

Unser Fachgebiet „Informatik & Gesellschaft“ an der Technischen Universität Berlin befasst sich in einem Arbeitsschwerpunkt mit den ökonomischen und politischen Implikationen von Open Source Software. Wir stellen fest, dass sich die Forschung bei diesen Fragen zunehmend mit den „Grossen“ der ICT-Branche befasst. Leider fehlen diesbezüglich aber noch weitgehend empirische Daten.

Deshalb begrüsse ich sehr, dass mein externer Doktorand, Herr Urs Lerch (lic. rer. pol.) mit Wohnsitz in der Schweiz, beabsichtigt, anhand einer Fallstudie bei grossen Schweizer ICT-Unternehmen diese Lücke teilweise zu schliessen. Mittels Fragebogenerhebung und Interviews bei Software-Entwicklern innerhalb der untersuchten Firmen möchte er Antworten auf folgende Fragen finden:

- Wie gestaltet sich die Arbeit von Open-Source-Entwicklern innerhalb grosser ICT-Firmen? Gibt es Unterschiede zur „Closed-Source“-Entwicklung?
- Welche Spannungsfelder ergeben sich zwischen der Open-Source-Community und grossen ICT-Firmen einerseits sowie zwischen Open-Source- und Closed-Source-Entwicklung innerhalb der Firmen andererseits?
- Wie beeinflussen sich Open-Source-Community und grosse ICT-Firmen?
- Welche Faktoren nehmen Einfluss auf die Open-Source-Tätigkeit in grossen ICT-Firmen? Gibt es Unterschiede zwischen unterschiedlichen Open-Source-Projekten innerhalb der Firmen?

Wir sind uns bewusst, dass eine solche Studie Zeitressourcen Ihrerseits beansprucht. Dennoch trete ich mit der Bitte an Sie heran, Herrn Lerch die Fallstudie zu ermöglichen. Dies dient sowohl dem Interesse der Wissenschaft

*Franklinstr. 28/29
Tel: +49 30 51475-420*

*D-12159 Berlin
Bernd@Lutterbeck.org*

Seite 2

als auch der weiteren Entwicklung von Open Source in der Schweiz und international. Es freut uns sehr, wenn wir mit Ihnen ins Gespräch kommen können bezüglich der Doktorarbeit von Herrn Lerch oder – wenn Sie dies wünschen – anderer für Sie interessanter Fragen zu Open Source.

Für weitere Auskünfte stehen Ihnen Herr Lerch (urs.lerch@tiscali.ch, 062 534 76 56) oder ich gerne zur Verfügung.

Im übrigen hat sich der bekannte englische Schriftsteller Glyn Moody dahin geäußert, dass ich „perhaps the most important collection of writings on open source and related areas to be found in any language“ herausgebe. Ich hoffe, er hat das Lob auch so gemeint. [www.opensourcejahrbuch.de]

Mit freundlichen Grüßen

Bernd Lutterbeck

B Vertraulichkeitserklärung

Anschrift

Vertraulichkeitserklärung

Im Rahmen meiner Dissertation am Lehrstuhl „Informatik & Gesellschaft“ an der Technischen Universität Berlin werde ich Daten anhand von Interviews in Ihrem Unternehmen erheben. Mir ist es bewusst, dass es sich dabei um sensible Informationen handeln kann. Deshalb sichere ich Ihnen strenge Vertraulichkeit beim Umgang mit diesen Daten zu.

Ich verpflichte mich, alle im Rahmen der Studie erhaltenen Daten und Informationen streng vertraulich zu behandeln, d. h. sie insbesondere nicht Dritten zugänglich zu machen und davon weder vollständige noch auszugsweise Kopien anzufertigen. Alle anfallenden Unterlagen werden spätestens bei Abnahme der Arbeit gelöscht. Ebenfalls verpflichte ich mich, Geschäftsgeheimnisse und Firmen-Know-How, welche mir zugänglich zu Kenntnis gekommen sind, streng vertraulich zu behandeln.

Die Ergebnisse der Befragungen werden in anonymisierter Form in meiner Dissertationsschrift publiziert. Wo die Anonymisierung nicht möglich oder sinnvoll erscheint, wird nur in Rücksprache und im Einverständnis mit Ihnen eine besser geeignete Form gewählt. Im Zweifelsfall muss auf die Verwendung der entsprechenden Informationen verzichtet werden.

Aarau, im September 2007

Adresse

Non Disclosure Statement

As part of my PhD at the institute "Science & Society" at the Technical University of Berlin I will be using data from interviews conducted in your company. I am aware that this might contain sensitive information. Therefore, in dealing with this data I will assure strict confidentiality during the study conduct and also after the study has ended.

I commit that all the material and data in the context of this study will be treated in strict confidence. In particular, the information will not be accessible to third parties. Neither do I make full nor partial photocopies or digital copies. I will keep all data material in my private home in a locked cupboard.

The study results will be anonymized for publication. Where anonymity is not possible or appropriate, I choose a more suitable form in consultation and in agreement with you. In case of doubt the use of such information will be omitted.

Aarau, September 2007

C Leitfaden

1. Leitfrage:

Damit ich Ihre heutige berufliche Situation verstehen kann, möchte ich zu Beginn unseres Gesprächs gerne wissen, wie denn so Ihre bisherige berufliche Entwicklung aussieht. [Herr/Frau ...] mich interessiert alles, was Ihren beruflichen Weg geprägt hat, und wie es dazu gekommen ist, dass Sie für [Firma] am Linux Kernel entwickeln? Bitte erzählen Sie, vielleicht angefangen in Ihrer Jugend, als sich die Frage der Berufswahl stellte.

Inhaltliche Aspekte	Aufrechterhaltungsfragen	Konkrete (Nach-)Fragen
<ul style="list-style-type: none"> - beruflicher Werdegang - Ausbildung - Beweggründe - interne/externe Besetzung 	<ul style="list-style-type: none"> - Und sonst? - Und weiter? - Und dann? - Was kam danach? 	<ol style="list-style-type: none"> 1. Welche Ausbildung haben Sie abgeschlossen? 2. Was hatten Sie denn noch für andere Jobs? 3. Was waren für Sie die wichtigsten Beweggründe, in den Informatikberuf einzusteigen? 4. Wie hat Sie [Firma] für die Arbeit an Linux gewinnen können?

2. Leitfrage:

Um nun konkret zum Thema „Alltag eines Open Source Entwicklers“ zu kommen: Erzählen Sie mir doch bitte, wie denn so ein typischer Arbeitsalltag bei Ihnen aussieht?

Inhaltliche Aspekte	Aufrechterhaltungsfragen	Konkrete (Nach-)Fragen
<ul style="list-style-type: none"> - Aufgaben - Routinetätigkeiten - Vorgehen - Interaktion und Kommunikation - Führung und Kontrolle - Verpflichtung 	<ul style="list-style-type: none"> - Und sonst? - Und weiter? - Und was ist sonst noch speziell? 	<ol style="list-style-type: none"> 1. Was sind die konkreten Aufgaben, die Sie zu erledigen haben? 2. Wie organisieren Sie sich in Ihrer Arbeit? 3. Wer sind Ihre hauptsächliche Bezugspersonen in Ihrer Arbeit? 4. Auf welche Art kommunizieren Sie (Mail, Telefon, Face-to-Face) denn am liebsten, was ist am praktischsten? 5. Wie sind Sie organisatorisch in der [Firma] eingebunden? 6. Sind Sie in der Community als [Name] der [Firma] bekannt?

3. Leitfrage:

Inwiefern unterscheidet sich in Ihrer Erfahrung die Arbeit an Open Source zur Closed Source Entwicklung?

Inhaltliche Aspekte	Aufrechterhaltungsfragen	Konkrete (Nach-)Fragen
<ul style="list-style-type: none"> - fachliche Anforderungen - soziale Anforderungen - Interaktion und Kommunikation - Führung und Kontrolle 	<ul style="list-style-type: none"> - Und sonst? - Und weiter? - Fällt Ihnen sonst noch etwas ein? 	<ol style="list-style-type: none"> 1. Was ist denn speziell an der Open-Source-Entwicklung? 2. Was für spezielle Fähigkeiten benötigen Sie? 3. War das für Sie eine Umstellung und wie haben Sie die bewältigt? 4. Wenn Sie das, was Sie machen, aus den Augen von anderen betrachten, was glauben Sie, was andere über Ihre Arbeit denken? 5. Wenn ein Kollege von Ihnen vor der Entscheidung stünde, Open Source oder Closed Source zu entwickeln, wie würden Sie ihn beraten?

4. Leitfrage:

Welche spezifischen Probleme entstehen für Sie durch die Arbeit an Open Source Software in einer kommerziell orientierten Firma?

Inhaltliche Aspekte	Aufrechterhaltungsfragen	Konkrete (Nach-)Fragen
<ul style="list-style-type: none"> - Organisation - soziale Einbindung - spezieller Arbeitsvertrag - rechtliche Absicherung - Umgang mit IP - psychologische Verträge - Leistungsbeurteilung - interne Konkurrenzprodukte - intrinsisch/extrinsische Motivation - Verbundenheit zur Community 	<ul style="list-style-type: none"> - Und sonst? - Und weiter? - Was fehlt Ihnen? - Was unternehmen Sie dagegen? - Wie könnte man Sie noch besser unterstützen? 	<ol style="list-style-type: none"> 1. Wie fühlen Sie sich von der Firma unterstützt? 2. Wie fühlen Sie sich in der Community integriert? 3. Bestehen Konflikte zu „internen“ Produkten oder Projekten? 4. Wie gehen Sie mit Konflikten von [Firma] mit der Community um? 5. Wenn Sie völlig frei entscheiden könnten, was würden Sie ändern, um Ihre persönliche Situation in Bezug auf Ihre Arbeit zu verbessern?

5. Leitfrage:

Inwiefern hat sich Ihre persönliche Einstellung zur Arbeit durch die Mitwirkung an Linux geändert?

Inhaltliche Aspekte	Aufrechterhaltungsfragen	Konkrete (Nach-)Fragen
<ul style="list-style-type: none"> - Einstellung zur Berufsarbeit - Einstellung zur Gemeinnützigkeit - Lohn, Ruhm - Verbundenheit zur Firma - Arbeitszufriedenheit 	<ul style="list-style-type: none"> - Und sonst? - Und weiter? 	<ol style="list-style-type: none"> 1. Hat sich die Bedeutung Ihres Berufs für Sie verändert? 2. Sie arbeiten ja jetzt mit Leuten sozusagen Hand in Hand, die das im Gegensatz zu Ihnen unentgeltlich machen. Hat sich dadurch Ihre Einstellung gegenüber Freiwilligenarbeit verändert? Und hat das denn auch praktische Auswirkungen? 3. Wie positionieren Sie sich in der Debatte von Freier versus Offener Software?

6. Leitfrage:

Ich würde jetzt noch gerne Ihre berufliche Situation etwas näher betrachten.
Wie würden Sie Ihre persönliche berufliche Situation heute charakterisieren?

Inhaltliche Aspekte	Aufrechterhaltungsfragen	Konkrete (Nach-)Fragen
<ul style="list-style-type: none"> - Zufriedenheit - Positives - Negatives - Berufsbild - Belastung - Work-Life-Balance - Perspektiven 	<ul style="list-style-type: none"> - Und sonst? - Und weiter? - Wo sehen Sie Probleme? 	<ol style="list-style-type: none"> 1. Sind Sie mit Ihrer aktuellen Situation zufrieden oder eher nicht? 2. Was sind die besonders positiven Aspekte Ihrer aktuellen Stelle? 3. Was sind die besonders negativen Aspekte Ihrer aktuellen Stelle? 4. Wann fühlen Sie sich denn besonders belastet? Was ist spannend an Ihrer Arbeit? Wie finden Sie Entspannung? 5. Wie vereinbaren Sie denn Berufs- und Privatleben? 6. Ziehen Sie es vor im Büro oder zuhause zu arbeiten? Warum? 7. Was ist denn für Sie ein „typischer Informatiker“? Sind Sie einer? 8. Was haben Sie für berufliche Zukunftsperspektiven?

Abschlussfrage:

Zum Abschluss unseres Gesprächs noch eine vielleicht etwas utopische Frage:
Wie würde denn ihr Traumberuf aussehen? Und wenn wir noch etwas weiter träumen, wie würde denn Arbeit aus Ihrer Sicht idealerweise (für die Gesellschaft) aussehen?

Herzlichen Dank für das Gespräch!

1. Guiding question:

As a beginning I would like to know a little bit about your current professional situation and your professional career.

I am interested in everything that your career has shaped. Would you please tell me, maybe starting in your youth, when you were confronted with the question of your career choice?

main issues	general questions	specific questions
<ul style="list-style-type: none"> - professional career - education - motivation - internal/external recruitment 	<ul style="list-style-type: none"> - What else? - And then? - What came next? 	<ol style="list-style-type: none"> 1. What's your education? 2. What were your jobs before the actual one? 3. What were your main motivations to choose the profession of a software developer? 4. How did it come that you develop on the Linux kernel for <i>[Firma]</i>?

2. Guiding question:

To be a bit more specific:

Could you please tell me, how a „typical“ working day of yours looks like?

main issues	general questions	specific questions
<ul style="list-style-type: none"> - tasks - routine activities - procedures - interaction and communication - management and control - commitment 	<ul style="list-style-type: none"> - What else? - And then? - And what else is special? 	<ol style="list-style-type: none"> 1. What are the specific tasks you have to do? 2. How do you organize yourself in your work? 3. Who are your main references in your work? 4. How do you preferably communicate (mail, phone, face-to-face), what is the most practical? 5. How are you organized in your company? 6. How are the Linux activities integrated in the group? 7. In the community, are you known as person or as employee of <i>[Firma]</i>? 8. Are you working in the office or at home?

<p>3. Guiding question: In your experience and opinion, how does working on Open Source software differ to the closed source development?</p>		
main issues	general questions	specific questions
<ul style="list-style-type: none"> - technical skills - social skills - communication - management and control 	<ul style="list-style-type: none"> - What else? - And then? 	<ol style="list-style-type: none"> 1. What is special about the Open Source software development? 2. What skills are needed compared to the closed source development? 3. Was it a change for you and how did you cope? 4. Are there special training courses? 5. Is the management of Open Source teams different? 6. What do you believe do closed source developers think about your work? 7. When a colleague of you had to decide to develop open source or closed source, what would be your advice?

<p>4. Guiding question: What are the specific problems created for you by the work on Open Source software in a commercially oriented company?</p>		
main issues	general questions	specific questions
<ul style="list-style-type: none"> - organization - social integration - labor contract - legal protection - IP - psychological contracts - measure of performance - internal products as competitors - intrinsic/extrinsic motivation - commitment to the community 	<ul style="list-style-type: none"> - What else? - And then? - What is missing? - What could the company do to give you a better support? 	<ol style="list-style-type: none"> 1. How is the support of the company? 2. Do you think you are integrated in the community? 3. Are there any conflicts between internal products and supported Open Source projects? 4. How are you treating conflicts between [<i>Firm</i>] and the community? 5. How important are legal aspects for you? 6. If you had three wishes, what would you change about your personal situation with respect to your work?

5. Guiding question:

Contributing to free software and collaborating in an open community, how has this affected your personal attitude to work?

main issues	general questions	specific questions
<ul style="list-style-type: none"> - attitude to professional work - attitude to the nonprofit activities - wage, fame - attachment to the company - job satisfaction 	<ul style="list-style-type: none"> - And then? - What else? 	<ol style="list-style-type: none"> 1. Did the importance of your job to you change? 2. So now you are working with people, as it were, hand in hand, in contrast to you free of charge. Has your attitude towards this volunteer work changed? And are there any practical implications for this reason? 3. What is your personal attitude concerning the debate about Free vs. Open Source software?

6. Guiding question:

I would like to take a closer look on your actual job situation.
How would you characterize your personal job situation today?

main issues	general questions	specific questions
<ul style="list-style-type: none"> - satisfaction - positive aspects - negative aspects - attitude to the profession - burden - work-life-balance - perspectives 	<ul style="list-style-type: none"> - And then? - What else? - Where do you see problems? 	<ol style="list-style-type: none"> 1. Are you satisfied with your current situation or not so much? 2. What are the most positive aspects of your current job? 3. What are the most negative aspects of your current job? 4. When do you feel especially burden? What is relaxing in your job? How do you relax? 5. How do you arrange your work-life-balance? Do you work in the evening at home? 6. Do you prefer to work in the office or at home? Why so? 7. In your eyes, what is a „typical“ software engineer? Are you typical? 8. What are your plans and prospects in your career?

Final question:

At the end of our conversation, perhaps even a bit an utopian question:
What would your dream job look like? And if we continue to dream a little bit, what should work look like best (for the society)?

Thank you very much for this conversation!

D Vorlage Datenblatt

Interviewprotokoll

Code:

Datum:
Ort:
Dauer:

Befragter:
Alter:
Ausbildung:
Firma:
Abteilung:
Anstellung seit:
Linux seit:
Status:

Interviewatmosphäre

zur Person

Einprägsame Aussagen

E Mail an unbekannte Beitragende

Von: Urs Lerch <lerch@cs.tu-berlin.de>
An:
Betreff: study concerning firms contribution in the linux kernel
Datum:

Hi,

As a PhD student at the Technical University of Berlin (Germany) I'm analyzing the logs of the linux kernel traffic. As I'm focusing on the firms contribution, I'm very much interested in who's employed by whom. As a matter of fact it is not that easy to do the matching and I would appreciate it a lot if you could help me with my study. You would do me a great favour if you could respond to the following questions:

- (1) Are you doing your contributions to the linux kernel
 - paid by a for-profit-organisation
 - paid by a non-profit-organisation
 - unpaid as academic
 - unpaid (non-academic)
- (2) What's the name of your organization?
- (3) Do I have to keep the data private?

I won't analyze the contributions by person but by the companies as a whole and will do some comparisons. So you're name or mailaddress won't be written in my study.

Thanks a lot!

Kind regards,

Urs

F Auswertung Linux-Kernel-Logs

	Entwicklung	Wartung
Autoren	2.097	374
Patches	28.334	1.335
Codezeilen	1.735.797	12.164
Personenjahre¹⁸⁸	1.062	4
Monetarisierung¹⁸⁹	79.650.000	300.000

Tabelle F.11: Übersicht der Beiträge am Linux Kernel im Jahr 2007

	v2.6.18	v2.6.19	v2.6.20	v2.6.21	v2.6.22	v2.6.23
Autoren	21	91	216	86	161	114
Patches	26	155	447	162	324	221
Codezeilen	230	1.653	4.167	1.573	2.550	1.991

Tabelle F.12: Übersicht der Beiträge an den einzelnen stable Releases in der **Wartung**

	Autoren		Patches		Codezeilen	
kommerziell	1.156	55,13 %	19.560	69,03 %	1.269.201	73,12 %
öffentlich	153	7,30 %	1.575	5,56 %	96.908	5,58 %
privat	410	19,55 %	4.988	17,60 %	269.048	15,50 %
unbekannt	378	18,03 %	2.211	7,80 %	100.640	5,80 %

Tabelle F.13: Gruppierung nach Interessen

	Personenjahre	Monetarisierung	Durchschnittliche Codezeilen pro Autor
kommerziell	748	56.099.027	1.098
öffentlich	42	3.145.749	633
privat	132	9.872.124	656
unbekannt	44	3.281.742	266

Tabelle F.14: Personenjahre, Monetarisierung und Aktivität nach Interessengruppen

188 Für die Berechnung der Personenjahre wurde das weit verbreitete, wenn auch etwas vereinfachende COCOMO-Modell (Boehm 2000) mit den Standardwerten für mittelkomplexe Software verwendet. Die Formel dafür lautet: **Personenjahre = 3 * Tausend Codezeilen¹¹² / 12**.

189 Für die Ermittlung des monetären Wertes wurde die Basis der MERIT-Studie (Ghosh 2006, S. 49) hinzugezogen, welche als Jahreskosten für ein Personenjahr € 75.000 verwendet, welches als vorsichtig bezeichnet wird. Die Monetarisierung wird somit auf der Basis von europäischen Werten vorgenommen, was bei der weltweiten Beteiligung nur einen informativen Wert haben kann.

	Autoren		Patches		Codezeilen	
kommerziell	226	60,43 %	941	70,49 %	9.037	74,29 %
öffentlich	28	7,49 %	126	9,44 %	977	8,03 %
privat	70	18,72 %	177	13,26 %	1.481	12,18 %
unbekannt	50	13,37 %	91	6,82 %	669	5,50 %

*Tabelle F.15: Interessengruppen in der **Wartung***

	Autoren		Patches		Codezeilen	
kommerziell	1.226	56,47 %	45.269	70,91 %	4.523.775	73,39 %
öffentlich	143	6,59 %	7.861	12,31 %	546.466	8,87 %
privat	357	16,44 %	7.721	12,09 %	775.032	12,57 %
unbekannt	445	20,50 %	2.989	4,68 %	318.528	5,17 %

Tabelle F.16: Gruppierung nach Interessen der Sign-offs

	Autoren		Patches		Codezeilen	
kommerziell	275	62,79 %	3.221	80,32 %	39.605	79,79 %
öffentlich	29	6,62 %	318	7,93 %	3.952	7,96 %
privat	73	16,67 %	344	8,58 %	4.282	8,63 %
unbekannt	61	13,93 %	127	3,17 %	1.797	3,62 %

*Tabelle F.17: Interessengruppen der Sign-offs in der **Wartung***

Kontinent	Firmen		Patches		Codezeilen	
Nordamerika	163	44,41 %	15.804	80,87 %	1.036.542	81,70 %
Europa	142	38,69 %	2.185	11,18 %	135.402	10,67 %
Asien	47	12,81 %	1.454	7,44 %	9.027	7,10 %
Afrika	3	0,82 %	84	0,43 %	6.691	0,53 %
Ozeanien	7	1,91 %	10	0,05 %	125	0,01 %
Südamerika	5	1,36 %	5	0,03 %	47	0,00 %

Tabelle F.18: Konsolidierung der Firmen in Kontinenten

Land	Firmen		Patches		Codezeilen	
USA	145	39,51 %	15.632	79,99 %	1.019.797	80,38 %
Japan	20	5,45 %	1.307	6,69 %	76.288	6,01 %
Deutschland	46	12,53 %	1.076	5,51 %	56.099	4,42 %
Großbritannien	21	5,72 %	565	2,89 %	35.436	2,79 %
Kanada	18	4,90 %	172	0,88 %	16.745	1,32 %
Finnland	4	1,09 %	124	0,63 %	15.806	1,25 %
Italien	6	1,63 %	24	0,12 %	7.551	0,60 %
Frankreich	15	4,09 %	160	0,82 %	7.540	0,59 %
Taiwan	9	2,45 %	35	0,18 %	7.240	0,57 %
Südafrika	3	0,82 %	84	0,43 %	6.691	0,53 %
Island	1	0,27 %	44	0,23 %	6.193	0,49 %
Israel	3	0,82 %	29	0,15 %	3.365	0,27 %
Schweden	10	2,72 %	57	0,29 %	2.957	0,23 %
Korea	1	0,27 %	32	0,16 %	1.437	0,11 %
Belgien	3	0,82 %	31	0,16 %	914	0,07 %
Indien	4	0,09 %	38	0,19 %	830	0,07 %
Polen	5	1,36 %	21	0,11 %	795	0,06 %
Spanien	5	1,36 %	8	0,04 %	716	0,06 %
Singapur	2	0,54 %	2	0,01 %	675	0,05 %
Tschechien	1	0,27 %	14	0,07 %	454	0,04 %
Niederlande	4	1,09 %	7	0,04 %	293	0,02 %
Ungarn	3	0,82 %	5	0,03 %	160	0,01 %
Schweiz	3	0,82 %	14	0,07 %	128	0,01 %
Australien	7	1,91 %	10	0,05 %	125	0,01 %
Österreich	5	1,36 %	9	0,05 %	124	0,01 %
Russland	6	1,63 %	7	0,04 %	82	0,01 %
Estland	1	0,27 %	4	0,02 %	80	0,01 %
China	1	0,27 %	3	0,02 %	62	0,00 %
Zypern	1	0,27 %	1	0,01 %	49	0,00 %
Brasilien	3	0,82 %	3	0,02 %	40	0,00 %
Dänemark	2	0,54 %	3	0,02 %	27	0,00 %
Irland	2	0,54 %	3	0,02 %	16	0,00 %
Weißrussland	1	0,27 %	4	0,02 %	15	0,00 %
Türkei	1	0,27 %	8	0,04 %	8	0,00 %
Argentinien	2	0,54 %	2	0,01 %	7	0,00 %
Serbien	1	0,27 %	1	0,01 %	4	0,00 %
Slowenien	1	0,27 %	2	0,01 %	3	0,00 %
Litauen	1	0,27 %	1	0,01 %	1	0,00 %

Tabelle F.19: Länderbeteiligung

Größe	Firmen		Autoren		Patches		Codezeilen	
Sehr groß	56	15,26 %	496	42,94 %	7.991	40,89 %	611.124	48,17 %
groß	42	11,44 %	255	22,08 %	6.593	33,74 %	385.572	30,39 %
mittel	70	19,07 %	153	13,25 %	1.651	8,45 %	65.871	5,19 %
klein	69	18,80 %	90	7,79 %	1.405	7,19 %	79.268	6,25 %
Sehr klein	130	35,42 %	161	13,94 %	1.902	9,73 %	126.918	10,00 %

Tabelle F.20: Gruppierung nach Firmengröße

Größe	Firmen		Autoren		Patches		Codezeilen	
Hardware	113	30,79 %	421	36,45 %	5.858	29,98 %	541.795	42,70 %
SW/Service	210	57,22 %	652	56,45 %	12.390	63,40 %	674.132	53,13 %
Telekomm	29	7,90 %	66	5,71 %	1.184	6,06 %	45.979	3,62 %
Handel	10	2,72 %	11	0,95 %	94	0,48 %	5.603	0,44 %
Non-IT	5	1,36 %	5	0,43 %	16	0,08 %	1.244	0,10 %

Tabelle F.21: Gruppierung nach Sektoren

Firma	Codezeilen		Patches		Autoren	
Red Hat	185.518	10,69 %	2.741	9,67 %	90	4,29 %
Intel	126.062	7,26 %	923	3,26 %	65	3,10 %
IBM	123.173	7,10 %	2.293	8,09 %	144	6,87 %
Analog Devices	101.074	5,82 %	357	1,26 %	11	0,52 %
Novell	91.195	5,25 %	2.161	7,63 %	56	2,67 %
SGI	44.646	2,57 %	1.074	3,79 %	30	1,43 %
Freescale	31.772	1,83 %	427	1,51 %	30	1,43 %
linutronix	31.215	1,80 %	506	1,79 %	5	0,24 %
Oracle	27.229	1,57 %	672	2,37 %	19	0,91 %
Renesas Technology	22.300	1,28 %	397	1,40 %	5	0,24 %
Sony	20.470	1,18 %	197	0,70 %	8	0,38 %
Chelsio	19.765	1,14 %	65	0,23 %	1	0,05 %
XenSource	19.348	1,11 %	127	0,45 %	3	0,14 %
MontaVista	19.203	1,11 %	306	1,08 %	26	1,24 %
Emulex	16.087	0,93 %	35	0,12 %	1	0,05 %
Atmel	14.555	0,84 %	137	0,48 %	7	0,33 %
Xmission	13.757	0,79 %	254	0,90 %	3	0,14 %
Cisco	12.825	0,74 %	116	0,41 %	4	0,19 %
Google	12.072	0,70 %	439	1,55 %	25	1,19 %
SBC	11.968	0,69 %	254	0,90 %	1	0,05 %
Broadcom	11.886	0,68 %	129	0,46 %	4	0,19 %
Simtec	11.225	0,65 %	137	0,48 %	2	0,10 %
Solid Boot Ltd	10.866	0,63 %	34	0,12 %	4	0,19 %
Qlogic	10.014	0,58 %	213	0,75 %	18	0,86 %
HP	9.959	0,57 %	230	0,81 %	24	1,14 %
Open Grid Computing	9.595	0,55 %	35	0,12 %	2	0,10 %
Wolfson Microelectr.	8.917	0,51 %	53	0,19 %	3	0,14 %
Toshiba	8.880	0,51 %	94	0,33 %	4	0,19 %
NetApp	8.875	0,51 %	275	0,97 %	4	0,19 %
SWsoft	8.871	0,51 %	382	1,35 %	19	0,91 %
Addtoit	8.386	0,48 %	226	0,80 %	1	0,05 %
Marvell	7.819	0,45 %	87	0,31 %	8	0,38 %
Astaro	7.763	0,45 %	276	0,97 %	1	0,05 %
SANPeople	6.499	0,37 %	62	0,22 %	1	0,05 %
LSI Logic	6.392	0,37 %	64	0,23 %	5	0,24 %
AMD	6.364	0,37 %	60	0,21 %	14	0,67 %
PMC-Sierra	6.221	0,36 %	7	0,02 %	1	0,05 %
OpenHand	6.193	0,36 %	44	0,16 %	1	0,05 %
Rowland Institute	5.907	0,34 %	169	0,60 %	1	0,05 %
rPath	5.905	0,34 %	75	0,26 %	1	0,05 %

Tabelle F.22: Top 40 Firmen nach beigetragenen Codezeilen

	Anzahl Entwickler in Top 30	Anteil am Total der Entwickler der Firma	Codezeilen pro Entwickler und Firma
Privatpersonen	5	1,22 %	656
Red Hat	5	5,62 %	2.084
Novell	3	5,36 %	1.628
Intel	3	4,62 %	1.939
linutronix	2	40,00 %	6.243
Simtec	1	50,00 %	5.613
Broadcom	1	25,00 %	2.972
Oracle	1	5,26 %	1.433
SBC	1	100,00 %	11.968
Cisco	1	25,00 %	3.206
Emulex	1	100,00 %	16.087
XenSource	1	33,33 %	6.449
SGI	1	3,33 %	1.488
Chelsio	1	100,00 %	19.765
Linux Foundation	1	16,67 %	8.834
IBM	1	0,69 %	855
Analog Devices	1	9,09 %	9.189

Tabelle F.23: Firmenzugehörigkeit der Top 30 Entwickler

Autoren pro Firma	Anzahl Firmen		Autoren		Patches		Codezeilen	
>51	4	1,08 %	355	30,74 %	8.118	41,57 %	525.948	41,45 %
21-50	5	1,34 %	135	11,69 %	2.476	12,68 %	117.652	9,27 %
11-20	7	1,88 %	111	9,61 %	1.856	9,50 %	159.835	12,60 %
6-10	11	2,96 %	81	7,01 %	902	4,62 %	61.572	4,85 %
5	9	2,42 %	45	3,90 %	1.286	6,59 %	69.892	5,51 %
4	9	2,42 %	36	3,12 %	735	3,76 %	58.209	4,59 %
3	22	5,91 %	66	5,71 %	903	4,62 %	58.791	4,63 %
2	30	8,06 %	60	5,19 %	653	3,34 %	36.743	2,90 %
1	272	73,12 %	272	23,55 %	2.605	13,34 %	180.111	14,20 %

Tabelle F.24: Gruppierung der Firmen nach Anzahl Autoren

	core	drivers	archi tecture	network	file systems	misc
kommerziell	71.790	643.965	364.674	77.506	79.224	32.042
öffentlich	5.992	64.673	11.241	7.981	3.457	3.564
privat	7.814	205.645	23.703	13.255	9.169	9.462
unbekannt	3.856	67.296	17.684	6.497	1.759	3.548

Tabelle F.25: Total Codezeilen pro Modul und Interessengruppe

	core	drivers	archi tecture	network	file systems	misc
kommerziell	80,26 %	65,61 %	87,39 %	73,65 %	84,63 %	65,91 %
Öffentlich	6,70 %	6,59 %	2,69 %	7,58 %	3,69 %	7,33 %
privat	8,74 %	20,95 %	5,68 %	12,60 %	9,79 %	19,46 %
unbekannt	4,31 %	6,86 %	4,24 %	6,17 %	1,88 %	7,30 %

Tabelle F.26: Anteil Codezeilen pro Interessengruppe und Modul

	core	drivers	archi tecture	network	file systems	misc
kommerziell	5,66 %	50,74 %	28,73 %	6,11 %	6,24 %	2,52 %
öffentlich	6,18 %	66,74 %	11,60 %	8,24 %	3,57 %	3,68 %
privat	2,90 %	76,43 %	8,81 %	4,93 %	3,41 %	3,52 %
unbekannt	3,83 %	66,87 %	17,57 %	6,46 %	1,75 %	3,53 %

Tabelle F.27: Codezeilen pro Modul in Relation zum Total der Interessengruppe

	core	drivers	archi tecture	network	file systems	misc
Nordamerika	60.589	518.220	292.189	63.558	77.585	24.401
Europa	8.541	84.927	24.127	11.791	1.059	4.860
Asien	2.596	39.409	43.031	2.157	533	2.302
Südamerika	2	45	0	0	0	0
Ozeanien	62	1.256	5.324	0	18	31
Afrika	0	93	3	0	29	0

Tabelle F.28: Total Codezeilen pro Modul und Kontinent

	core	drivers	archi tecture	network	file systems	misc
Sehr groß	28.697	282.548	225.530	21.029	34.912	18.408
Groß	29.935	201.715	64.274	40.936	40.023	8.239
Mittel	4.066	27.475	21.322	10.482	2.024	502
Klein	3.486	49.533	20.808	2.233	1.477	1.731
Sehr klein	5.606	82.694	32.290	2.826	788	2.714

Tabelle F.29: Total Codezeilen pro Modul und Firmengröße

	core	drivers	archi tecture	network	file systems	misc
Hardware	15.165	284.267	208.937	8.807	12.256	12.363
SW/Dienstl.	47.221	330.327	150.281	66.413	63.260	16.630
Telekomm	9.259	23.049	5.081	2.284	3.708	2.598
Handel	43	5.199	359	2	0	0
Non-IT	102	1.123	16	0	0	3

Tabelle F.30: Total Codezeilen pro Modul und Sektor

Architektur	kommerziell		öffentlich		privat		unbekannt	
386	25.116	89,91 %	1.167	4,18 %	1.356	4,85 %	297	1,06 %
x86	11.724	76,73 %	852	5,58 %	2.428	15,89 %	276	1,81 %
AMD64	8.372	91,77 %	532	5,83 %	179	1,96 %	40	0,44 %
IA-64	7.720	94,81 %	130	1,60 %	235	2,89 %	58	0,71 %
s390	19.690	98,48 %	67	0,34 %	178	0,89 %	58	0,29 %
PowerPC	83.361	89,29 %	491	0,53 %	4.838	5,18 %	4.670	5,00 %
Sparc	22.261	93,07 %	78	0,33 %	946	3,96 %	634	2,65 %
Alpha	425	29,43 %	411	28,46 %	138	9,56 %	470	32,55 %
PA-RISC	6.676	79,80 %	59	0,71 %	745	8,91 %	886	10,59 %
MIPS	27.756	63,46 %	3.851	8,80 %	8.226	18,81 %	3.906	8,93 %
ARM	38.795	80,98 %	2.322	4,85 %	3.982	8,31 %	2.810	5,87 %
H8/300	132	16,42 %	4	0,50 %	30	3,73 %	638	79,35 %
Blackfin	91.415	99,91 %	10	0,01 %	35	0,04 %	37	0,04 %
SuperH	19.004	78,58 %	215	0,89 %	2.754	11,39 %	2.210	9,14 %
AVR32	6.423	98,97 %	26	0,40 %	32	0,49 %	9	0,14 %
CRIS	1.721	91,11 %	8	0,42 %	145	7,68 %	15	0,79 %
FR-V	295	65,85 %	86	19,20 %	25	5,58 %	42	9,38 %
M32R	5.970	98,78 %	18	0,30 %	44	0,73 %	12	0,20 %
M68K	710	26,17 %	781	28,79 %	373	13,75 %	849	31,29 %
V850	75	63,65 %	2	1,69 %	35	29,66 %	6	5,08 %
Xtensa	1.962	96,84 %	29	1,43 %	21	1,04 %	14	0,69 %

Tabelle F.31: Architekturen nach Interessengruppe in Codezeilen und Prozent

Architektur	Firma mit den meisten Beiträgen		Firma mit den zweitmeisten Beiträgen		Firma mit den drittmeisten Beiträgen	
386	XenSource		rPath		Novell	
	7.984	28,58 %	4.955	17,74 %	2.753	9,85 %
x86	linutronix		SGI		XenSource	
	5.513	36,08 %	1.179	7,72 %	886	5,80 %
AMD64	IBM		Novell		Google	
	2.752	30,17 %	1.819	19,94 %	859	9,42 %
IA-64	Intel		SGI		Fujitsu	
	3.207	39,38 %	1.627	19,98 %	910	11,18 %
s390	IBM		Red Hat		Novell	
	19.259	96,33 %	104	0,52 %	79	0,40 %
PowerPC	IBM		Freescale		Sony	
	31.362	33,59 %	21.278	22,79 %	9.281	9,94 %
Sparc	Red Hat		Oracle		Novell	
	21.821	91,23 %	128	0,54 %	42	0,18 %
Alpha	HP		Novell		Analog Devices	
	250	17,31 %	72	4,99 %	19	1,32 %
PA-RISC	Red Hat		Swoff		Intel	
	6.378	76,24 %	82	0,98 %	75	0,90 %
MIPS	SGI		PMC-Sierra		Tripeaks	
	16.242	37,13 %	5.197	11,88 %	3.177	7,26 %
ARM	SANPeople		Simtec		Marvell	
	5.343	11,15 %	4.980	10,39 %	4.328	9,03 %
H8/300	Analog Devices		Red Hat		Novell	
	59	7,34 %	24	2,99 %	13	1,62 %
Blackfin	Analog Devices		HV Systemas		Red Hat	
	91.077	99,54 %	263	0,29 %	29	0,03 %
SuperH	Renesas		igel		MSC	
	10.062	41,61 %	4.569	18,89 %	1.542	6,38 %
AVR32	Atmel		SBC		SGI	
	5.848	90,11 %	430	6,63 %	35	0,54 %
CRIS	Axis		Novell		Red Hat	
	1.557	82,42 %	41	2,17 %	36	1,91 %
FR-V	Red Hat		Xmission		SGI	
	99	22,10 %	72	16,07 %	30	6,70 %
M32R	Renesas		Red Hat		Novell	
	5.868	97,09 %	24	0,40 %	22	0,36 %
M68K	Snapgear		Novell		Red Hat	
	549	20,24 %	26	0,96 %	24	0,88 %
V850	Red Hat		Novell		linutronix	
	23	19,49 %	9	7,63 %	9	7,63 %
Xtensa	Tensilica		Red Hat		Novell	
	1.875	92,55 %	26	1,28 %	21	1,04 %

Tabelle F.32: Firmen mit den meisten Beiträgen pro Architektur

	Codezeilen Architektur	Anteil Firma an Total Architektur	Anteil Architektur an Total Firma
Analog Devices	91.949	21,12 %	90,97 %
IBM	55.337	12,71 %	44,93 %
Red Hat	30.689	7,05 %	16,55 %
SGI	21.532	4,95 %	48,23 %
Freescale	21.290	4,89 %	67,01 %
Renesas Technology	16.185	3,72 %	72,58 %
Monta Vista	11.690	2,68 %	60,88 %
Sony	9.281	2,13 %	45,34 %
Intel	9.031	2,07 %	7,16 %
XenSource	8.943	2,05 %	46,22 %
linutronix	8.266	1,90 %	26,48 %
Novell	6.354	1,46 %	6,97 %
Atmel	5.854	1,34 %	40,22 %
Toshiba	5.459	1,25 %	61,48 %
rPath	5.396	1,24 %	92,38 %
SANPeople	5.347	1,23 %	82,27 %
PMC-Sierra	5.197	1,19 %	83,54 %
Simtec	4.980	1,14 %	44,37 %
igel	4.569	1,05 %	99,96 %
Marvel	4.328	0,99 %	55,35 %
rmk	3.493	0,80 %	69,35 %
Tripeaks	3.179	0,73 %	80,30 %
Vmware	2.731	0,63 %	97,40 %
CompuLab	2.575	0,59 %	81,38 %
SBC	2.225	0,51 %	18,59 %
SecretLab	2.187	0,50 %	48,66 %
Tensilica	1.891	0,43 %	100,00 %
Lemote	1.840	0,42 %	100,00 %
Addtoit	1.756	0,40 %	20,94 %
Axis Communications	1.557	0,36 %	99,24 %

Tabelle F.33: Top 30 Firmen im Architekturbereich

G Verwendete Software

Da sich die vorliegende Arbeit mit dem Phänomen „Open Source“ beschäftigt, war es naheliegend, ausschließlich mit Open-Source-Software zu arbeiten. Dieser Anspruch hat sich als nicht ganz einfach herausgestellt.

Im Bereich des wissenschaftlichen Arbeitens ist die Menge an freien Programmen nicht besonders groß. Zumindest konnte nur wenige oder für das Vorhaben nur bedingt geeignete Software recherchiert werden. Die Gründe dafür sind unklar. Zum einen scheinen die Budgets (noch) groß genug zu sein, sodass sich die Institute die kommerziellen Lizenzen leisten können. Zum anderen ist die Sensibilität, eigens erstellte Software unter eine Open-Source-Lizenz zu stellen, noch gering. Mit der Diskussion um Open Access könnte sich dies jedoch ändern.

Die Linux-Distribution Ubuntu und OpenOffice¹⁹⁰ sollen hier nicht näher erläutert werden, da sie mittlerweile zum Mainstream gezählt werden können. Hingegen wird nachfolgend die verwendete Software kurz beschreiben, die noch wenig bekannt ist.

– Zotero (Bibliographie-Software)

Zu Beginn wurde noch die in Java programmierte Anwendung JabRef¹⁹¹ verwendet. Die Oberfläche ist intuitiv zu gebrauchen und bietet alle notwendige Funktionalität. Die Daten werden in einer Textdatei im BibTex-Format abgelegt und sind unabhängig von der Software zu verwenden. Das große Problem von JabRef ist jedoch, dass es nicht in OpenOffice integrierbar und deshalb eine Referenzierung im Dokument nicht (direkt) möglich ist. Aus diesem Grund wurde mit dem Beginn der Niederschrift zu Zotero¹⁹² gewechselt, welches zu diesem Zeitpunkt mit der ersten Betaversion in Erscheinung getreten ist. Mittlerweile ist es weitverbreitet und hat aus Sicht des Verfassers das Potenzial, sich zukünftig zum Quasi-Standard zu entwickeln.

190 Als Schrifttyp wurde der freie Font „Free Serif“ (<http://www.gnu.org/software/freefont/> [05.09.2009]) verwendet, der unter der GNU GLP Version 3 steht.

191 Siehe <http://jabref.sourceforge.net/> [10.08.2009]. JabRef ist im Repository von Ubuntu zumindest ab der Version 8.04 enthalten.

192 Siehe <http://www.zotero.org/> [10.08.2009]. Zotero steht unter der „Educational Community Licence“, die von der OSI unterstützt wird.

Die Migration war einfach über die Import-Funktion von Zotero möglich.¹⁹³ Die Verwaltung der Literaturdatenbank von Zotero läuft über ein Plug-in im Browser Firefox (siehe Abb. G.39) problemlos, bietet jedoch etwas weniger Komfort als JabRef. Die Datenbank wird lokal in einer SQLite-Datei abgelegt, kann jedoch mittlerweile auch zentral auf einem Server verwaltet werden.

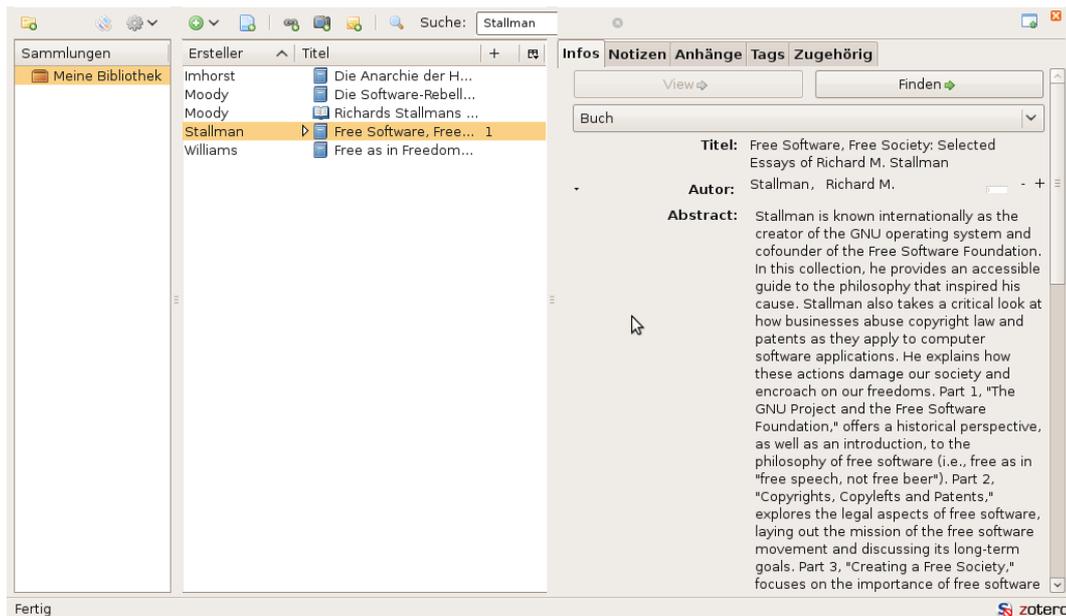


Abbildung G.39: Screenshot Zotero

Zotero bietet ein Plug-in für OpenOffice – und auch Microsoft Word –, durch das die Referenzierung im Dokument ermöglicht wird. Die Funktionalität wird im Menü sowie in einer Toolbar angeboten, Shortcuts fehlen (soweit bekannt) noch. Die Referenzierung funktioniert allerdings nur, wenn der Browser geöffnet und eine Internetverbindung vorhanden bzw. das Offline-Tag im Menü „Datei“ von Firefox deaktiviert ist. Unter gewissen nicht genau eruierten Bedingungen besteht eine Limitierung der Referenzen in einem Dokument; aufgrund von diversen Foreneinträgen liegt diese bei ungefähr 200 Literatureinträgen. Zudem war es in einzelnen Verweisen notwendig, manuell Korrekturen anzubringen, da das Plugin nicht immer korrekt den Verweistext generiert hat. Es existieren zahlreiche Stylesheets für die Referenzierung, die im XML-Format gehalten sind und deshalb einfach angepasst und ergänzt werden können. Für die vorliegende Arbeit wurde dasjenige

193 Allerdings waren die von den beiden Programmen verwendeten BibTeX-Formate nicht vollständig identisch. Aus diesem Grunde wurde ein kleines Perl-Script geschrieben, das insbesondere die Felder für Autoren und Herausgeber in mehrere Einzelattribute splittet.

von Harvard übernommen und leicht modifiziert.

– Transcriber (Transkriptions-Software)

Für die Transkription erwies sich Transcriber¹⁹⁴, „ein Werkzeug für das Segmentieren, Labeln und Transkribieren gesprochener Sprache“, als äußerst praktisch. Die Software wurde mit der Scriptingsprache Tcl/Tk erstellt, steht unter der GNU-Lizenz und läuft unter Linux, Mac und Windows. Verwendet wurde die Version 1.5.1, die sich auch im Universe-Repository von Ubuntu (zumindest ab Version 07.10) befindet. Für das zweite Quartal 2009 ist eine vollständig neu entwickelte Version angekündigt, die jedoch zum Zeitpunkt der Fertigstellung dieser Dissertation noch nicht erschienen ist.

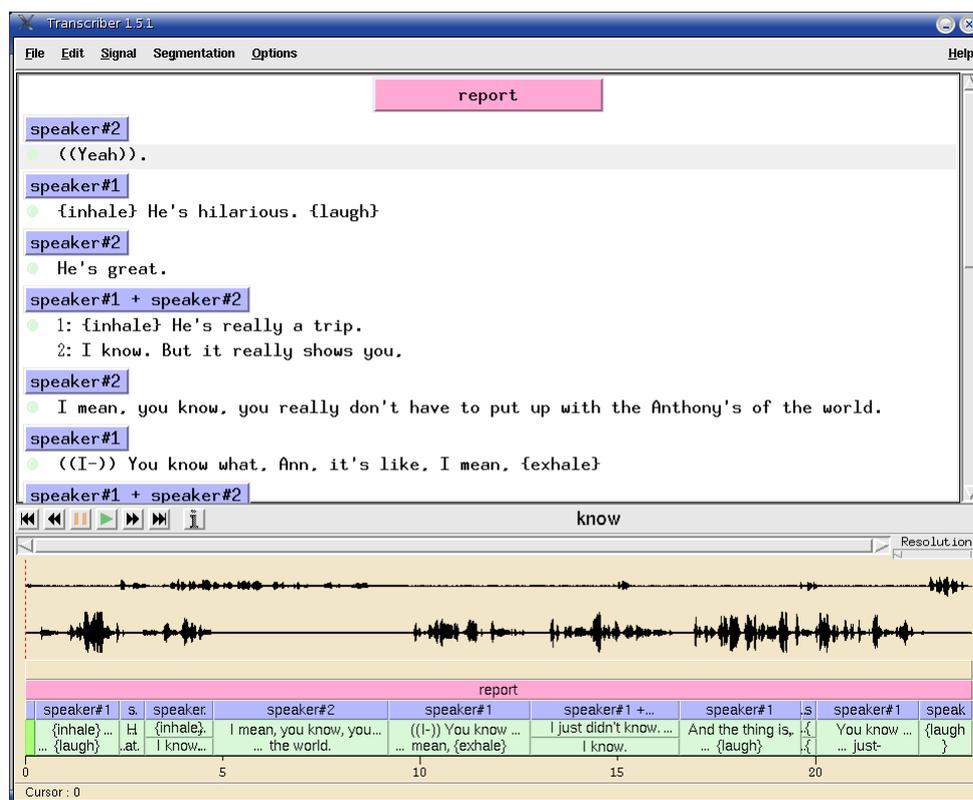


Abbildung G.40: Screenshot Transcriber

Nutzerfreundlich bei Transcriber ist, dass es sich um einen Editor mit integriertem Player handelt (siehe Abb. G.40). Dabei wird bei jedem neuen Segment ein Zeitpointer festgehalten, sodass beim Navigieren im Transkript auch die Audiodatei „mitscrollt“, was sich als sehr praktisch herausstellte. Beim Transkribieren lassen sich alle notwendigen Operatio-

194 Siehe <http://trans.sourceforge.net/en/presentation.php> [17.08.2009].

nen einfach mit der Tastatur ausführen. Besonders hilfreich ist die Einstellung, dass beim Anhalten der Pointer um eine frei definierbare Sekundenanzahl zurückspringt, so dass man nie zurück spulen muss. Das Transkript wird als XML-Datei abgelegt und kann in verschiedene Formate, unter anderem HTML und ASCII-Text, exportiert werden. Die Weiterverwendung in anderen Programmen, zum Beispiel zum Codieren, stellt folglich kein Problem dar.

– Weft QDA (Qualitative Analyse-Software)

Das Codieren der transkribierten Texte ist zwar in Transcriber möglich. Die Funktionalität ist allerdings derart rudimentär, dass dies nicht zu empfehlen ist. Auch das auf Linux lauffähige GTAMS Analyzer¹⁹⁵ überzeugt insbesondere in Bezug auf Benutzerfreundlichkeit nicht. Als das einzige vielversprechende Open-Source-Werkzeug zur Codierung von Texten wurde WeftQDA¹⁹⁶ verwendet. Einzuschränken ist, dass die Software unter Linux nicht ganz einfach zu installieren ist. Alternativ kann sie zwar mittels der Wine-Emulation verwendet werden, weist dort aber einige unerklärliche und stark störende Bugs auf, so dass das Programm in einer Windows-Virtual-Box verwendet wurde.

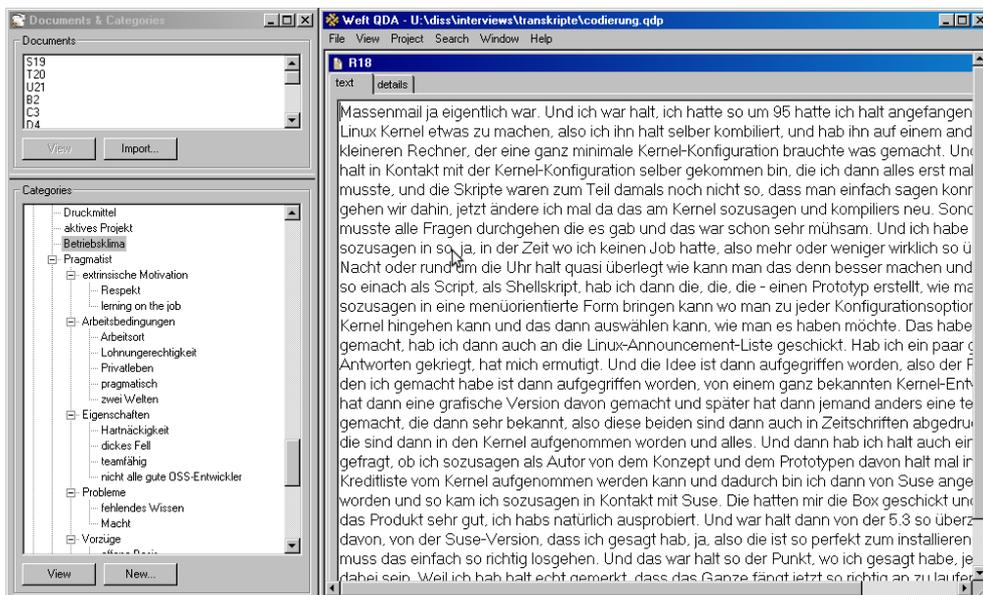


Abbildung G.41: Screenshot Weft QDA

195 Siehe <http://tamsys.sourceforge.net/gtams/> [17.08.2009]. Das Programm ist im Repository von Ubuntu, zumindest ab Version 7.04 enthalten.

196 Siehe <http://www.pressure.to/qda/> [17.08.2009]. WeftQDA ist grundsätzlich lizenziert als Public Domain, wobei einzelne Bibliotheken und Komponenten einer anderen Lizenz entsprechen können.

Die Arbeitsumgebung ist dreigeteilt: Eine Liste der zu codierenden Texte sowie die in einer Baumstruktur organisierbaren Codes bilden die linke Navigationsseite, auf der rechten Seite wird der Transkriptionstext angezeigt (siehe Abb. G.41). Eine Textstelle kann einfach codiert werden, indem sie markiert wird und anschließend in einer Listbox am unteren Rand des Fensters der Code eingegeben wird. Das Hinzufügen von neuen Codes muss etwas umständlich im Codebaum auf der linken Seite zuerst eingefügt werden, bevor er zugeteilt werden kann. Das größte Manko ist jedoch, dass immer nur die Textstellen eines Codes im Textfenster markiert und somit eine nachträgliche Bearbeitung kaum möglich ist. Die nachträgliche Organisation des Codebaumes ist auch viel zu umständlich gelöst. Als Konsequenz daraus wurde nach erfolgtem Codieren, wofür das Programm durchaus zu gebrauchen ist, die resultierenden Daten in eine Tabelle exportiert und dort weiter bearbeitet. Die kommerziellen Programme (insbesondere MaxQDA und Atlas-TI) sind den Open-Source-Varianten aufgrund ihrer langjährigen Erfahrung immer noch weit überlegen, und es bietet sich an, diese zu verwenden. Allerdings wäre es aus Sicht des Verfassers ein sinnvolles Hochschulprojekt, eine Open-Source-Lösung für die qualitative Analyse von Transkripten zu starten.