Kooperation in der Open Source Entwicklung

Eine empirische Untersuchung des Debian Projekts

Lizentiatsarbeit am Institut für Soziologie
Wirtschafts- und Sozialwissenschaftliche Fakultät
der Universität Bern

13. Februar 2009

verfasst von:
Gaudenz Steinlin
Langmauerweg 112
3011 Bern
96-112-081

eingereicht bei:
PD Thomas Gautschi
Institut für Soziologie
Lerchenweg 36
3012 Bern

Danksagung

Ich danke Stefan Wehrli für seine zahlreichen Anregungen und kritischen Kommentare zu meiner Forschungsarbeit. Philipp Ackermann danke ich für die Durchsicht des Theoriekapitels. Franziska Grossenbacher für die oftmals nötigen Motivationsspritzen und für das Korrekturlesen. Zudem bedanke ich mich bei den Informatikdiensten der Universität Bern dafür, dass sie mir für mein Projekt zur Datenbearbeitung Zugang zum Linux-Cluster «Ubelix» gewährten und bei der Professur für Soziologie der ETH Zürich für den Zugang zu ihrem Linux Rechner «sociolix» für die statistische Auswertung der Daten. Nicht zu vergessen sind alle Mitglieder des Debian Projekts, welche durch ihren Einsatz meine Arbeit erst möglich gemacht haben.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

«Just for fun: the story of an accidental revolutionary.» (Torvalds & Diamond 2001) heisst die Autobiographie des in einer breiteren Öffentlichkeit bekanntesten Open Source Entwicklers Linus Torvalds. Entsteht Open Source Software einfach nur aus Spass? Warum entwickeln tausende von hochqualifizierten Softwareentwicklern und einige wenige Softwareentwicklerinnen[1] qualitativ hochstehende Programme und verschenken diese gratis an alle Interessierten? Was für Linus Torvalds als «Spassprojekt» begann, erlaubt ihm heute, seine Familie zu ernähren. Und nicht nur er, sondern unzählige weitere Open Source Entwickler weltweit werden heute dafür bezahlt, täglich Software zu entwickeln und sie inklusive dem, traditionellerweise hoch geheimen, Programmcode an die ganze Welt zu verschenken. Ist es reiner Altruismus, der diese Menschen dazu bringt, sich an Open Source Projekten zu beteiligen? In keinem anderen Industriezweig hat Altruismus in der Vergangenheit jedoch eine bedeutende Rolle gespielt. Warum sollten also Softwareentwickler besonders altruistische Menschen sein. Lässt sich so die Entstehung von Softwarepaketen wie des Internet-Browsers Firefox und der Office-Suite OpenOffice erklären? Beides Open Source Produkte, welche erfolgreich mit kommerziellen Alternativen aus dem Hause Microsoft konkurrenzieren.

Am Anfang der «Revolution» steht Richard Stallman. Der Ursprungsmythos der Freien Software[2] handelt davon, dass Stallman der Zugang zum Quellcode eines Druckertreibers verweigert wurde, den er um eine für ihn sehr nützliche Funktion erweitern wollte. Zu

[1] Ich werde im weiteren im Interesse einer einfachen Sprache ausschliesslich die männliche Schreibweise verwenden. Leider wird Open Source Software fast ausschliesslich von Männern entwickelt (siehe dazu auch Abschnitt 2.1.1). Ich kann deshalb auch wegen der zu kleinen Fallzahl keine Aussagen über Entwicklerinnen machen. In den letzten Jahren hat die Anzahl der Entwicklerinnen insbesondere innerhalb des Debian Projektes zugenommen. Dies ist auch auf Projekte wie Debian Women `http://women.debian.org/` zurückzuführen. Es bleibt zu hoffen, dass sich in den nächsten Jahren vermehrt auch Frauen an der Open Source Entwicklung beteiligen, so dass ich dann mit dieser billigen Entschuldigung nicht mehr durchkomme.

[2] Innerhalb der Gemeinschaft der «Free, Libre and Open Source Software» Bewegung gibt es eine grössere Kontroverse um die richtige Bezeichnung. Die von Stallman angeführte Free Software Foundation propagiert den Begriff «freie Software», da damit die zentrale Komponente der Freiheit betont werden soll. Als Gegenpol dazu wurde die «Open Source Initiative» gegründet, die Open Source Software insbesondere auch Unternehmen schmackhaft machen möchte. Sie lehnt die Bezeichnung «free» ab, da sie nicht nur mit Freiheit, sondern im Englischen auch gratis bedeutet. Die Bezeichnung «Open Source» dagegen betone das wesentliche, nämlich die Verfügbarkeit des Quellcodes. Daneben wurde von der Europäischen Kommission auch der Begriff «Libre Software» zur Vermeidung dieser Zweideutigkeit eingeführt. Dieser hat sich jedoch nicht durchgesetzt. Oft wird zur Vermeidung eines Positionsbezugs auch die Abkürzung FOSS (Free Open Source Software oder gar FLOSS (Free Libre Open Source Software) verwendet. In praktischer Hinsicht sind diese Kontroversen jedoch unbedeutend. Sowohl die Proponenten der freien Software, wie auch jene des Begriffs Open Source verstehen darunter bis auf geringfügige Details das Gleiche. Ich werde im weiteren in meiner Arbeit den Begriff Open Source verwenden, da es mir nicht um eine Analyse der Ideale der Bewegung geht, sondern um die konkrete

diesem Zeitpunkt, in den frühen 1970er Jahren, war es üblich, dass zu einem Programm auch der Quellcode mitgeliefert wurde. Software wurde damals noch vornehmlich als Zugabe zur sehr teuren Hardware verstanden. Die Erfahrung, dass sich Hackerkollegen weigerten, den Quellcode ihrer Programme mit anderen zu teilen und die Tatsache, dass Firmen, welche proprietäre Software ohne Quellcode verkauften, immer dominanter wurden, führte Stallman dazu, in den frühen 1980er Jahren das GNU Projekt zu gründen. Ziel dieses Projektes ist es, ein komplett freies Betriebssystem zu entwickeln (Williams 2002). Bei Stallman stand also zu Beginn sein eigenes Interesse an einer Programmfunktion im Vordergrund. Die meisten Open Source Projekte beginnen mit einem einzigen oder wenigen Entwicklern, die ein Programm in ihrem eigenen Interesse entwickeln.

Viele Projekte verbleiben in diesem Stadium. Bei anhaltendem Interesse der ursprünglichen Entwickler, können so auch erfolgreiche Projekte entstehen. Für die soziologische Analyse interessanter sind jedoch Projekte, denen es gelingt, über diese Initialzündung hinaus einen grösseren Kreis an Entwicklern anzuziehen. Für diese steht nicht mehr zwingend der direkte Nutzen der Entwicklung im Vordergrund. Ich versuche in meiner Arbeit zu erklären, dass auch diese Entwickler nicht einfach altruistisch handeln, sondern dass sich ihre Verhalten auch unter der Annahme rationaler Akteure erklären lässt.

Open Source Software ist ein «kollektives Gut». Die ökonomische Theorie geht klassischerweise davon aus, dass ein solches Gut nicht oder zumindest in suboptimalem Ausmass produziert wird. Da das Gut nicht nur den Produzenten, sondern allen zur Verfügung steht, wird die Mehrheit versuchen, sich nicht an der Produktion zu beteiligen und als «Free Rider» nur vom Produkt zu profitieren. Wie lässt sich unter diesen Voraussetzungen die Entwicklung von Open Source Software erklären? Zur Verschärfung der Problematik trägt bei, dass die Beteiligten eines Open Source Projekts sich meist nicht persönlich kennen und die Entwicklung im weitgehend anonymen Raum des Internets stattfindet. Wie kommt Kooperation von einander fremden Entwicklern in der Anonymität des Internet zustande?

Die Produktion kollektiver Güter und damit die Entwicklung von Open Source Software ist eine klassische Dilemmasituation. Wenn sich alle an der Entwicklung beteiligen, dann stehen alle am Ende besser da. Für jeden einzelnen Entwickler ist es jedoch vorteilhafter, sich nicht zu beteiligen und nur von der Entwicklung der anderen zu profitieren. Dies ist das Verhalten des «Free Riders». In der Spieltheorie werden Dilemmasituationen oft mit Hilfe des Gefangenendilemmas betrachtet. Für das Gefangenendilemma ist bekannt, dass in iterierten Spielen kooperative Strategien erfolgreich sein können. In meiner Arbeit werde ich untersuchen, ob dies auch für die Open Source Entwicklung eine mögliche Erklärung ist.

Ökonomen erklären die Produktion kollektiver Güter hingegen oft durch selektive Anreize, von welchen die «Free Rider» nicht profitieren können (Olson 1965). Ein solcher Anreiz ist, dass sich Entwickler über die Beteiligung an Open Source Projekten eine Reputation als fähige Softwareentwickler aufbauen. Diese Reputation können sie beispielsweise zur Akquisition lukrativerer Aufträge einsetzen. Doch lohnt sich der Aufbau einer Reputation auch innerhalb eines Open Source Projekts? Trägt sie dazu bei, dass andere Entwickler sie

Kooperation im praktischen Projektalltag.

als kooperationswillige Projektmitglieder erkennen und deshalb eher bereit sind, ihrerseits mit ihnen zu kooperieren?

Diesen Fragen zur, auf den ersten Blick eher unwahrscheinlich erscheinenden, Kooperation unter Open Source Entwicklern werde ich in meiner Arbeit nachgehen. Mit Hilfe empirischer Daten aus dem Debian Projekt möchte ich zeigen, dass sich ein guter Ruf lohnt und dass er auch innerhalb eines Open Source Projekts gewinnbringend genutzt werden kann. Ich versuche zu zeigen, wie Kooperation innerhalb dieses Projekts entsteht, aber auch, in welchen Fällen sie scheitert. Meines Wissens gibt es bisher noch keine Studie, welche die Wirkung von Reputation in einem Open Source Projekt an empirischen Daten untersucht hat.

Für den empirischen Teil meiner Arbeit werde ich Daten des Debian Projekts[3] benutzen. Das Debian Projekt stellt die gleichnamige Linux Distribution her. Eine Distribution ist die Form, in der ein Linux Betriebssystem üblicherweise installiert wird. In einer Distribution werden Open Source Programme aus verschiedensten Quellen gebündelt und so bereitgestellt, dass sie auf einfache Art und Weise installiert werden können. Die Aufgabe der Debian Entwickler ist nicht primär die Entwicklung von Software, sondern das Zusammenstellen der Programme der verschiedensten Open Source Projekte zu einem für Anwender brauchbaren Betriebssystem. Die meisten Linux Benutzer installieren Open Source Software in der Form von Softwarepaketen, welche von der Distribution zur Verfügung gestellt werden. Eine Distribution ist das «Bindeglied» zwischen den einzelnen Open Source Projekten und den Endbenutzern. Eine Distribution erzeugt Zusatznutzen für die Benutzer, da diese eine zentrale Bezugsquelle und einen zentralen Ansprechpartner haben. Aber auch die einzelnen Projekte profitieren von den Distributionen dadurch, dass sie vom Endbenutzersupport entlastet werden. Neben Debian bestehen noch zahlreiche weitere Distributionen.[4]

Das Debian Projekt ist eines der grössten Open Source Projekte. Es umfasst mehr als 1'000 offizielle Entwickler und ein Vielfaches an weiteren Interessierten, die sich in irgendeiner Weise am Projekt beteiligen. Debian ist die bei Open Source Entwicklern beliebteste Distribution (Ghosh et al. 2002: 29). Durch seine Grösse und weil ein beträchtlicher Teil der Benutzer selbst Open Source Entwickler in anderen Projekten sind, eignet sich das Projekt sehr gut für die Analyse. Kooperation in einer solch grossen Gruppe ist mit Sicherheit nicht trivial und damit bieten sich ideale Gelegenheiten, die Kooperationsmechanismen genauer zu untersuchen. Open Source Projekte zeichnen sich generell dadurch aus, dass ein Grossteil der Arbeit öffentlich stattfindet. Dies ist auch für das Debian Projekt nicht anders. Im «Social Contract», einem «Vertrag» mit der weiteren Open Source Community verpflichtet sich das Debian Projekt, alle Fehler in seiner Fehlerdatenbank jederzeit öffentlich zu behalten. Die Daten aus dem Debian Projekt sind deshalb relativ einfach und vollständig verfügbar. Daneben gibt es aber auch einen forschungspraktischen Grund für die Auswahl des Debian Projekts. Durch meine eigene

[3] http://www.debian.org/

[4] Die bekanntesten sind die beiden grossen kommerziellen Distributionen Red Hat und SuSE, sowie die neuere Distribution Ubuntu, welche auf Debian basiert.

Arbeit als Debian Entwickler, wenn auch in den letzten Jahren nicht mehr sehr intensiv, bin ich mit den Prozessen des Projekts vertraut. Dies erleichtert es, die riesige Fülle von verfügbaren Daten zu bewerten und für die Forschung nutzbar zu machen. Denn auch wenn die Daten der meisten Open Source Projekte öffentlich zugänglich sind, braucht es zu ihrer korrekten Interpretation oft ein nicht unwesentliches Detailwissen zur Funktionsweise eines Projekts.

Die Arbeit ist im folgenden in fünf weitere Kapitel gegliedert. Im Kapitel 2 werde ich den Forschungsgegenstand Open Source Software Entwicklung vorstellen. In den letzten Jahren hat sich Open Source Software von einem nur wenigen Experten bekannten Phänomen zu einem einer grösseren Allgemeinheit zumindest ansatzweise geläufigen Begriff gewandelt. Deshalb wird hier auf eine detaillierte Darstellung verzichtet. Parallel mit der Popularisierung der Open Source Bewegung hat sich auch die Wissenschaft begonnen, sich mit dem Phänomen auseinander zu setzen. Deshalb soll der aktuelle Forschungsstand kurz geschildert werden. Die Forschung ist jedoch mittlerweile so breit, dass eine komplette Darstellung eine eigene Arbeit füllen würde. Ich konzentriere mich deshalb auf diejenigen Untersuchungen, deren Ansätze und Ergebnisse für meine weitere Arbeit genutzt werden können. Abgeschlossen wird dieses Kapitel mit einem Überblick über die spezifischen Prozesse innerhalb des Debian Projekts, welche Gegenstand der empirischen Analyse sein werden.

Im Kapitel 3 werde ich ausgehend von der Spieltheorie ein Modell der Erstellung von Open Source Software als kollektives Gut entwickeln. Kollektive Güter wurden bereits in grosser Zahl spieltheoretisch untersucht. Jedoch ist mir keine Untersuchung bekannt, welche diese Konzepte umfassend auf die Entwicklung von Open Source Software angewandt hat. Die Erstellung eines kollektiven Gutes ist ein Paradebeispiel eines sozialen Dilemmas. Zentrale Aufgabe des Theorieteils ist es deshalb, zu klären, wie Kooperation unter rationalen Akteuren in einem Open Source Projekt möglich ist. Zum Schluss werden ausgehend vom entwickelten Modell empirisch prüfbare Hypothesen formuliert.

Bevor die Hypothesen geprüft werden können, werden im Kapitel 4 die erhobenen Daten zur Arbeit an der Debian Linux Distribution und die zu verwendenden statistischen Modelle erläutert. Insbesondere auf die spezifischen Problemen der Datenerhebung im Internet wird näher eingegangen. Aus den erhobenen Daten wurden zwei Datensätze gewonnen. Der eine erfasst die Arbeit der einzelnen Personen in Monatsintervallen und der andere die Arbeit an der Fehlerdatenbank der Debian Distribution. Die in den Datensätzen enthaltenen Daten werden kurz überblicksartig dargestellt. Die statistischen Verfahren werden dagegen nur kurz präsentiert. Es wird auf die Spezifika der einzelnen Verfahren eingegangen und erläutert, warum sie zur Prüfung der Hypothesen geeignet sind. Für ausführlichere Darstellungen muss jedoch auf die zahlreich vorhandenen Lehrmittel verwiesen werden.

Das Kapitel 5 bildet den Hauptteil der Arbeit. Hier werden zuerst einige deskriptive Ergebnisse aus dem Datensatz der Entwickleraktivität vorgestellt. Diese Ergebnisse dienen einerseits dazu, ein genaueres Bild von der Projektarbeit zu erhalten und andererseits können einige Ergebnisse von grösser angelegten Surveys über Open Source Entwickler

an den erhobenen Prozessdaten validiert werden. Danach werden die im Theoriekapitel aufgestellten Hypothesen am Datensatz der Fehlerberichte geprüft und ausführlich diskutiert.

In Kapitel 6 wird die Arbeit mit einer Zusammenfassung der wichtigsten Resultate und einigen weiterführenden Betrachtungen abgeschlossen.

2 Forschungsgegenstand

2.1 Forschungsstand zu Open Source Software

Seit dem Jahre 2000 hat das Forschungsinteresse am Phänomen Open Source Software rapide zugenommen. Seither haben sich zahlreiche Publikationen mit dem Phänomen der Open Source Software befasst und eine Vielzahl verschiedenster Erklärungsansätze für dieses präsentiert (für eine Übersicht vergleiche Feller et al. 2005). Während soziologische Untersuchungen im engeren Sinn immer noch eher selten sind, sind zahlreiche Untersuchungen auf den Gebieten der Informatik und informatiknaher Wissenschaften, der Management- und Organisationslehre und der Wirtschaftswissenschaften entstanden.

Wenige Autoren haben sich mit der Bedeutung von Status und Reputation für die Entwicklung von Open Source Software auseinandergesetzt. Stewart (2005) untersucht die Statusmobilität und -stabilität in der Open Source Community. Er benutzte dazu Daten von Advogato.org, einer Internet-Plattform, mit der einzelne Entwickler die Leistung ihrer Kollegen bewerten können. Diese Plattform kann als Vorläufer der heute beliebten social-network Plattformen (beispielsweise Facebook) betrachtet werden. Stewart untersucht die Dynamik der gegenseitigen Bewertungen. Er stellt dabei insbesondere fest, dass der Anteil reziproker Bewertungen erstaunlich hoch ist. Falls A eine Bewertung für B abgegeben hat, dann wird die Wahrscheinlichkeit, dass B auch A bewerten wird 60 mal grösser. Auch zeigt sich, dass die Statusbewertungen nach einiger Zeit abnehmen und der Status einer Person relativ stabil bleibt. Der Aufbau einer guten Reputation ist danach vor allem kurz nach dem Eintritt in die Community wichtig. Es geht Stewart darum, den Einfluss von Status und Reputation auf den Open Source Entwicklungsprozess zu untersuchen. Auch wenn die Fragestellung eine gewisse Ähnlichkeit zu meiner Arbeit aufweist, lassen sich die Ergebnisse nur bedingt vergleichen. Zu gross sind die Unterschiede zwischen seinem Datensatz und den Daten, die ich verwenden werde. Während in der vorliegenden Arbeit Daten untersucht werden, die direkt im Entwicklungsprozess entstanden sind, hat Stewart Daten verwendet, die nicht Prozessdaten zur Arbeit an einem Open Source Projekt sind. Seine Daten basieren auf Peer-Ratings. Brand & Holtgrewe (2004) untersuchen die Organisation des Open Source Projekts KDE. Sie befassen sich dabei auch mit der Bedeutung von Reputation. Dabei finden sie heraus, dass Reputation insbesondere der Regulation von Aufmerksamkeit dient. Die Reputation eines Autors wird als Filter für die Beachtung seiner Beiträge auf den Mailinglisten verwendet. So können die Entwickler in der riesigen Informationsflut die interessantesten Beiträge herausfiltern. Beim Review von eingesandtem Programmcode funktioniert die Reputation genau umgekehrt. Beiträge von Entwicklern mit noch geringer Reputation werden genauer unter die Lupe genommen, bevor sie dem Programm hinzugefügt werden.

Zum Debian Projekt gibt es bisher nur sehr wenig Forschung. Am umfassendsten ist wohl

die Arbeit von Gabriella Coleman (Coleman 2005, Coleman & Hill 2005). Sie beschäftigt sich aus einer anthropologischen Perspektive mit Debian. Ihre interessante Arbeit ist aber leider für meine Fragestellung nicht anschlussfähig. Sie beschäftigt sich insbesondere mit dem Aufnahmeprozess für offizielle Entwickler als «Eintrittsritual» und der damit verbunden Formierung eines «ethischen» Habitus. O'Mahony & Ferraro (2004) untersuchen die Auswirkung der Beteiligung an Mailinglistendiskussionen und des Kontaktes mit anderen Entwicklern auf die Wahlbeteiligung an der Projektleiterwahl. Sie zeigen, dass aktive und besser vernetzte Personen sich stärker für das Projekt engagieren und sich diese Engagement auch in einer höheren Beteiligungsquote in der Projektleiterwahl zeigt. Dieser Schluss scheint mir aber fast tautologisch. Welcher soziale Zusammenhang damit gezeigt werden soll, bleibt unklar. Interessant ist allenfalls der Einbezug der Unterschriften im Schlüsselbund für die elektronische Signatur um das Ausmass des «Offline»-Kontaktes zwischen Entwicklern zu messen.[1] Eine weitere, eher qualitative ausgerichtete, Untersuchung wurde von Garzarelli & Galoppini (2003) erstellt. Sie zeigen, dass es innerhalb des Debian Projekts trotz des sehr modularen Aufbaus der Distribution teilweise hierarchische Organisationsstrukturen gibt.

Es gibt einige wenige Versuche, Prozessdaten der Open Source Entwicklung zu analysieren (Mockus et al. 2002, Koch & Schneider 2000, Krishnamurthy 2002: beispielsweise). Diese beschränken sich aber meist auf mehr oder weniger deskriptive Darstellungen des Datensatzes. Bei diesen Autoren stehen nicht soziologische Fragestellungen im Vordergrund, sondern sie interessieren sich aus einer «Software Management» Perspektive für das Phänomen Open Source Software. Sie untersuchen die Codequalität und das Projektmanagement oder vergleichen den Softwareentwicklungsprozess mit proprietärer Softwareentwicklung. Mir ist keine Untersuchung bekannt, welche Prozessdaten aus einem Open Source Projekt unter soziologischen Gesichtspunkten analysiert. Genau dies soll jedoch mit der vorliegenden Arbeit versucht werden.

2.1.1 Open Source Entwickler

Im folgenden werden einige quantitative und mehrheitlich deskriptive Daten zu Open Source Entwicklern vorgestellt. Dabei geht es insbesondere darum, den meinem Datensatz inhärenten Nachteil etwas wettzumachen, dass ich über keine Informationen über die beteiligten Entwickler verfüge, welche über die aktuelle Projektarbeit hinausgehen.

Aus den zahlreichen (Online-) Befragungen von Open Source Entwicklern habe ich nur zwei ausgewählt, die genauer vorgestellt werden sollen. Die Zusammenfassung der Resultate gibt einen Überblick über die Entwickler. Die beiden grössten Studien sind unter den Namen FLOSS-POLS (Free/Libre/Open Source: Policy Support) (Ghosh et al. 2002) und FLOSS-US (David et al. 2003) bekannt. Obwohl keine der Untersuchungen sich explizit auf das Debian Projekt konzentrierte, ist anzunehmen, dass die Resultate mehrheitlich auch auf das Debian Projekt übertragbar sind.

Zwischen Juni 2001 und Mai 2002 wurde am International Institute of Infonomics

[1] Mit ihrer elektronischen Unterschrift bestätigt eine Person, dass sie die Identität des Inhabers des Schlüssels geprüft hat. Per Konvention darf eine Person dies nur tun, wenn sie bei einem persönlichen Treffen einen amtlichen Ausweis des Inhabers geprüft hat.

der Universität Maastricht eine von der Europäischen Union finanzierte, umfassende Studie zu Freier und Open Source Software (FLOSS-POLS) durchgeführt. Im Rahmen dieser Studie wurden 2784 Entwickler von Freier Software zu ihrer Tätigkeit befragt. Die Befragung wurde mit Hilfe eine Online Fragebogens durchgeführt, welcher in der Community möglichst breit gestreut wurde. Die Ergebnisse der Befragung sind im vierten Teil des Schlussberichts des FLOSS-POLS Projektes zusammengefasst (Ghosh et al. 2002). Die FLOSS-US Studie wurde 2003 am Stanford Institute for Economic Policy Research (SIEPR) mit einem fast identischen Fragebogen durchgeführt. Diesmal wurde der Fragebogen verstärkt auch unter nordamerikanischen Entwicklern gestreut, die in der FLOSS-POLS Studie untervertreten waren (David et al. 2003). Auch diese Befragung wurde Online durchgeführt. In beiden Studien konnte jedermann, den Fragebogen ausfüllen. Es wurde kein randomisiertes Sample gezogen. Deshalb sind die Ergebnisse mit einiger Vorsicht zu geniessen. Besser Daten existieren leider nicht.

Tabelle 2.1 gibt einen Überblick über die Ergebnisse der beiden Studien. Sie sind sehr ähnlich. Bedeutende Unterschiede gibt es nur beim Wohnsitz und beim Beschäftigungsstatus. Im folgenden sollen die im Bezug auf meine Fragestellung wichtigsten Erkenntnisse zusammengefasst werden.

Tabelle 2.1: Zusammenfassung der Ergebnisse aus FLOSS-POLS und FLOSS-US

Geschlecht	**POLS**	**US**
männlich	98.9%	98.4%
weiblich	1.1%	1.6%
Alter		
Median	26	27
Median beim Eintritt in die Open Source Entwicklung	22	22
Zivilstand/Partnerschaft		
Alleinstehend	41.4%	40.5%
Partnerschaft	58.6%	58.5%
davon verheiratet	21.1%	28.0%
eigene Kinder	17.0%	19.0%
Ausbildung		
Universitätsabschluss	70%	74.4%
davon Undergraduate/Bachelor	33%	36.3%
davon Graduate/Master oder PhD	27%	36.7%
Arbeit		
angestellt	65%	51.7%
selbständig	14%	15.9%
Student	17%	28.8%

Wohnsitz		
Westeuropa	70%	52.7%
Nordamerika	14%	27.1%
andere	16%	20.1%
Zeiteinsatz (pro Woche)		
weniger 10h	70%	
davon weniger als 2h	22.5%	
davon 2-5h	26.1%	
davon 6-10h	20.9%	
11-20h	14.3%	
21-40h	9.1%	
über 40h	7.1%	
Anzahl beteiligte Projekte		
1-2 Projekte	56%	
2-10 Projekte	34%	
mehr als 10	10%	
Monetäre Entschädigung (Mehrfachantworten möglich)		
keine Entschädigung	46.3%	
direkt, bezahlt für Systemadministration	17.5%	
direkt, bezahlt für Entwicklung	15.7%	
direkt, bezahlt für Support	11.9%	
indirekt, Stelle wegen FOSS Erfahrung erhalten	17.5%	
indirekt, FOSS Entwicklung auch am Arbeitsplatz	12.8%	
indirekt, aber FOSS Entwicklung ist nicht Teil des Pflichtenhefts	5.2%	

Anmerkung: Quellen: FLOSS-POLS:(Ghosh et al. 2002), FLOSS-US: (David et al. 2003); teilweise eigene Zusammenfassung und Berechnung der Prozentanteile; Daten zum Zeiteinsatz, zur Anzahl Projekte und zur monetären Entschädigung sind in FLOSS-US nicht enthalten

Der durchschnittliche Entwickler ist männlich, zwischen 20 und 30 Jahre alt, hat eine sehr gute Ausbildung und lebt in Nordamerika oder Westeuropa. Gut die Hälfte lebt in einer Partnerschaft. Nur wenige habe jedoch eigene Kinder, was zu einem Grossteil wohl durch das Alter der Entwickler erklärt werden kann.

Entwicklerinnen sind eine Rarität. Der Anteil der Frauen unter den Entwicklern ist sehr gering. Nur gut 1% sind weiblich. Unter den offiziellen Debian Entwicklern ist dieser Anteil sogar noch kleiner. Von 1181 offiziellen Entwicklern in meinem Datensatz sind nur gerade 11 Frauen (0.9%).[2] Dieser Frauenanteil ist wesentlich kleiner als in anderen Bereichen der Informatik. Zwar sind in der Informatik die Frauen generell untervertreten. Sie sind

[2]Die Zahl von 11 Frauen basiert auf einer Diskussion auf der Mailingliste debian-women. `http://lists.debian.org/debian-women/2008/01/msg00043.html`

jedoch im Studienfach Informatik an Universitäten wesentlich stärker vertreten als in der Open Source Entwicklung. In der Schweiz betrug der Frauenanteil an Universitäten im Fach Informatik 11%.[3]

Der Grossteil arbeitet entweder im IT Sektor oder studiert Informatik oder ein verwandtes Fach. Entgegen der verbreiteten Vorstellung, dass Open Source Software vor allem von Studenten in ihrer Freizeit programmiert wird, ist jedoch die Mehrheit der Entwickler angestellt oder selbständig erwerbend. Auch verdient eine knappe Mehrheit der Entwickler zumindest indirekt Geld durch ihr Engagement. Etwa 1/6 der Entwickler ist direkt für die Entwicklung von Open Source Software angestellt. Auffallend ist der grosse Anteil an Selbständigerwerbenden.

Die Arbeit in einem Open Source Projekt ist für die meisten Entwickler ein Teilzeitjob. Weniger als 10% wenden mehr als 40 Stunden pro Woche dafür auf. Dagegen engagieren sich 70% der Entwickler mit weniger als 10 Stunden. Der Zeiteinsatz variiert nicht gross mit dem Beschäftigungsstatus. Angestellte und Selbständige wenden durchschnittlich sogar etwas mehr Zeit auf. Dies spricht auch dafür, dass ein beträchtlicher Anteil von Open Source Software nicht als Hobby, sondern im Rahmen einer bezahlten Arbeit entwickelt wird. Gut die Hälfte der Entwickler sind nur in einem bis zwei Projekten aktiv. Nur knapp 10% sind in über 10 Projekten engagiert. Diese beiden Ergebnisse decken sich auch mit einer Untersuchung zum GNOME Projekt (Koch & Schneider 2000), welche durch eine Auswertung des Quellcodes herausgefunden hat, dass der grösste Teil des Programmcodes von einer relativ kleinen Zahl von Personen geschrieben wird.

Die Motivation zur Beteiligung an einem Open Source Software Projekt wurde in FLOSS-POLS und FLOSS-US mit zwei Fragen ermittelt. Einerseits wurden Faktoren abgefragt, die zum Einstieg in die Open Source Community beigetragen haben und andererseits wurde nach Gründen gefragt, in der Gemeinschaft zu verbleiben. Am häufigsten wurden Motivationsfaktoren genannt, welche den Lernaspekt betonen. So sind 78.9% zur Gemeinschaft gestossen, um «neue Fähigkeiten zu erlernen» und 67.2% bleiben in der Gemeinschaft, um Wissen und Fähigkeiten weiterzugeben. Für 30% bis 35% sind ideologische Gründe ausschlaggebend, wie die Ansicht, dass «Software nicht ein proprietäres Gut sein soll» und die «Beteiligung an einer neuen Form von Kooperation». Neben diesen beiden Bereichen werden aber auch reputationsbezogene Gründe direkt angesprochen: So geben 12% an, dass sie in der Gemeinschaft bleiben um sich eine gute Reputation aufzubauen und 29.8% möchten ihre Berufsaussichten verbessern. Die rein monetäre Faktoren scheinen am Anfang einen eher geringen Einfluss zu haben. So gaben nur 4.4% an, dass sie begonnen hätten, sich an einem Projekt zu beteiligen, weil sie Geld verdienen wollten. Dagegen verbleiben 12.3% weiterhin im Projekt um Geld zu verdienen. Ich werde auf die Motivation, sich an einem Open Source Projekt zu beteiligen, im Rahmen der Diskussion der Nutzenfunktion von Open Source Entwicklern im Abschnitt 3.2.1 nochmals zurückkommen.

[3] Quelle: Bundesamt für Statistik

2.2 Softwareentwicklung im Debian Projekt

Im Folgenden wird der Prozess der Softwareentwicklung im Debian Projekt kurz vorgestellt. Diese Erläuterungen dienen zum Verständnis der im weiteren untersuchten Vorgänge. Auf die Darstellung von technischen Details, welche für das Verständnis nicht von Bedeutung sind, wird jedoch verzichtet. Das Folgende ist deshalb kein vollständige Beschreibung des Entwicklungsprozesses.

Das Debian Projekt entwickelt eine sogenannte «Distribution» des Open Source Betriebssystems Linux. Neben Debian bestehen noch zahlreiche weitere Distributionen[4], welche untereinander um die Gunst der Linux Benutzer konkurrieren. Eine Linux Distribution besteht aus einzelnen Softwarepaketen, welche zusammen das Linux Betriebssystem bilden. Dabei werden nur die wenigsten der in den Pakten enthaltenen Programme speziell für eine Distribution entwickelt. Die Aufgabe einer Distribution ist nicht primär die Entwicklung von Software, sondern das Zusammenstellen der Programme der verschiedensten Open Source Projekte zu einem für Anwender brauchbaren Betriebssystem.

2.2.1 Entwicklungsprozess

Anders als die meisten Linux Distributionen wird Debian nicht von einer Firma entwickelt, sondern von nur lose verbundenen Individuen. Die meisten dieser Entwickler engagieren sich ohne Bezahlung als Freiwillige für das Projekt. Einige arbeiten jedoch auch als Teil ihres Erwerbsarbeit an Debian. Beispielsweise in einem Unternehmen, das Debian verwendet, oder an einer Universität. Debian als Organisation hat jedoch keine bezahlten Mitarbeitenden. Die Mitglieder des Debian Projekts sind über die ganze Welt verstreut. Sie koordinieren sich mit Hilfe der Internet-Kommunikation. Dabei werden vor allem Mailinglisten und Internet Chat verwendet. Zudem gibt es eine jährliche Konferenz (DebConf) und verschiedene ad-hoc Treffen, an denen sich die Entwickler auch «Face to Face» sehen.

Softwarepakete

Die einzelnen Softwarepakete werden meist von einem einzelnen Entwickler oder einem kleinen Team von Entwicklern erstellt und betreut. Der zuständige Paketbetreuer ist in jedem Paket vermerkt. Um eine neue Version eines Paketes zur Distribution hinzuzufügen, muss ein Entwickler dieses mit seinem kryptographischen Schlüssel signieren und auf einen Server des Projekts hochladen. Pakete können nur von offiziellen Entwicklern hochgeladen werden. Alle anderen Entwickler müssen ihre Pakete zuerst an einen offiziellen Entwickler senden, der sie prüft und dann hochlädt. Dieser Vorgang wird in Debian «sponsern eines Pakets» genannt.

[4] Die bekanntesten sind die beiden grossen kommerziellen Distributionen Red Hat und SuSE, sowie die neuere Distribution Ubuntu, welche auf Debian basiert.

Fehlerberichte

Das Debian Projekt unterhält eine Datenbank, in welcher jeder Benutzer des Betriebssystems Fehler melden kann.[5] Die Steuerung der Datenbank erfolgt über Email. Sie ist offen für alle und es wird keine Registrierung oder ähnliches benötigt, um einen Fehler zu melden oder zu verändern. Zu einem Fehler können wiederum per Email von Jedermann Zusatzinformationen hinzugefügt werden. Sobald ein Fehler behoben wurde, wird dies in der Datenbank vermerkt. Ein Fehler kann auf verschiedene Arten behoben werden. Die mit Abstand häufigste Variante ist, dass er automatisch nach dem hochladen eines Softwarepakets, welches den Fehler beseitigt, geschlossen wird. Ein Fehler kann jedoch auch per Email geschlossen werden, mit dem Tag «wontfix» markiert werden oder das betroffene Softwarepaket kann aus der Distribution entfernt werden.

2.2.2 Status von Debian Entwicklern

Im Folgenden sollen die Entwickler in vier hierarchisch geordnete Statuskategorien unterteilt werden. Diese Kategorien wurden ex-post aus dem Datensatz gebildet. Es handelt sich dabei also nicht direkt um Statuszuschreibungen, welche auch im Debian Projekt so verwendet werden. Eine Ausnahme davon ist der Status des offiziellen Entwicklers. Diesen erhält ein Entwickler erst, nachdem er einen relativ aufwändigen Aufnahmeprozess durchlaufen hat. Um diesen Prozess zu starten, muss ein Entwickler von einem anderen Entwickler, der bereits aufgenommen wurde, empfohlen werden. Im Rahmen dieses Prozesses müssen die Kandidaten beweisen, dass sie fähig sind, Softwarepakete zu erstellen, die den Projektrichtlinien und anerkannten Qualitätsstandards entsprechen. Es wird aber nicht nur die technische Kompetenz geprüft, sondern die Entwickler verpflichten sich auch, sich an die Grundsätze des Debian Projekts zu halten.[6] Ausserdem muss die Identität des Entwicklers von mindestens einem offiziellen Entwickler bestätigt werden. Der Aufnahmeprozess dauert im Schnitt 6 bis 12 Monate. Um als offizieller Entwickler aufgenommen zu werden, ist also eine beträchtliche Investition an Zeit und Engagement notwendig.

Auch wenn die weiteren in meiner Unterteilung verwendeten Statuskategorien nicht direkt in dieser Form innerhalb des Debian Projekts verwendet werden, gehe ich doch davon aus, dass viele Entwickler implizit eine ähnliche Einteilung vornehmen.

Contributor Diese Personen haben mindestens einen Eintrag in der Fehlerdatenbank. Die überwiegende Mehrheit dieser Personen hat mindestens einmal einen Fehler in einem Debian Softwarepaket gemeldet. Vereinzelte haben jedoch nur Zusatzinformationen zu einem bestehenden Fehler beigetragen oder einen Fehler als behoben markiert. Diese Personen werden wohl den meisten anderen Beteiligten unbekannt sein. Sie haben deshalb den tiefsten Status.

[5] Die Datenbank kann online unter http://bugs.debian.org/ eingesehen werden.

[6] Jeder offizielle Entwickler muss vor seiner Aufnahme die «Debian Free Software Guidelines» und den «Social Contract» digital signieren. In diesen beiden Dokumenten sind die Grundsätze des Debian Projekts festgehalten.

Einfache Entwickler Diese Personen haben sich mindestens einmal an der Entwicklung eines Softwarepakets beteiligt. Sie tauchen mit mindestens einem Eintrag in einem Änderungslog eines Pakets auf. Sie sind jedoch weder offizielle Entwickler noch gehören sie zur Gruppe der Kernentwickler. Den meisten Beteiligten werden auch diese Personen weitgehend unbekannt sein. Sie haben aber durch ihre Entwicklungsarbeit im Kreis der Entwickler, mit denen sie zusammen arbeiten, eine gewisse Bekanntheit erlangt. Auch kann von jedermann einfach überprüft werden, ob eine Person sich bereits an der Entwicklung beteiligt hat. Ihr Status ist deshalb höher als der eines Contributors.

Offizielle Entwickler Diese Personen können Softwarepakete in das Softwarearchiv des Debian Projekts hochladen. Ausserdem dürfen sie an Wahlen und Abstimmungen des Projekts teilnehmen. Sie haben den offiziellen Aufnahmeprozess durchlaufen. Nicht zu dieser Gruppe gehören Personen, welche auch zur Gruppe der Kernentwickler gehören. Offizielle Entwickler sind relativ einfach an ihrer Mailadresse zu erkennen. Auch wenn sie nicht die offizielle Adresse benutzen, kann ohne Schwierigkeiten überprüft werden, ob jemand ein offizieller Entwickler ist. Die meisten offiziellen Entwickler werden deshalb von ihren Interaktionspartner als solche erkannt.

Kernentwickler Diejenigen 20% der Entwickler, welche über den ganzen Untersuchungszeitraum die meiste Arbeit an Paketen geleistet haben. Die Kernentwickler leisten zusammen gut 80% der Arbeit. 95% der Kernentwickler sind auch offizielle Debian Entwickler. Durch ihre umfangreiche Arbeit werden diese Entwickler allen Personen, die sich nicht nur sehr sporadisch am Projekt beteiligen bekannt sein.

Die Zugehörigkeit zu einer dieser Kategorien bedeutet nicht, dass die betreffende Person nicht auch an der charakterisierenden Aktivität einer anderen Kategorie beteiligt wäre. So melden praktisch alle Entwickler auch Fehler. Eine Person wurde jeweils der derjenigen Kategorie zugeordnet, welche den höchsten Grad an Beteiligung am Projekt voraussetzt. Auch bauen die Kategorien aufeinander auf. So waren die meisten Kernentwickler wohl zu Beginn ihrer Beteiligung einfach Fehlerberichterstatter, und wurden nach einer gewissen Zeit als einfach Entwickler zu offiziellen Entwicklern und gehören nun durch ihre intensive Mitarbeit zur Gruppe der Kernentwickler.

Die Definition der Kategorie der Kernentwickler ist für alle Betrachtungen, welche den Datensatz in Zeitabschnitte aufteilen, problematisch, da sie auf dem gesamten Untersuchungszeitraum basiert und nicht nur auf dem betrachteten Zeitabschnitt. Dies führt dazu, dass es in den früheren Zeitabschnitten weniger Kernentwickler gibt. Jedoch ist auch eine Definition, welche die gesamte Arbeit jeweils nur auf den betrachteten Abschnitt bezieht nicht unproblematisch. Mit einer solchen Definition wäre der Kernentwicklerstatus der einzige, den eine Person auch wieder verlieren könnte. Dies würde die Modellierung erheblich komplexer machen. Ich habe mich deshalb entschieden, bei der einfacheren Definition zu bleiben. Ausserdem hat sich in den empirischen Ergebnissen gezeigt, dass die Unterschiede zwischen den jeweiligen Entwicklerkategorien relativ gering sind. Somit hat die exakte Definition für die Ergebnisse auch nur eine untergeordnete Bedeutung.

3 Theorie

Ziel dieses Teils meiner Arbeit ist es, ein theoretisches Modell der Entwicklung von Open Source Software zu entwerfen, um aus diesem Modell empirisch prüfbare Hypothesen abzuleiten. Es erstaunt, dass sich eine grosse Zahl von Entwicklern an Open Source Projekten beteiligt, obwohl dies auf den ersten Blick ihren rationalen Interessen zuwiderläuft. Diese Entwickler beteiligen sich freiwillig an Projekten und wenden dafür oft einen nicht unerheblichen Teil ihrer Zeit auf, obschon sie dafür keinerlei monetäre Entschädigung erhalten. Und dies nicht in einem traditionellen Bereich der Freiwilligenarbeit, sondern in einem Feld, in dem üblicherweise überdurchschnittliche Löhne für die gleiche Arbeit gezahlt werden.

Ich konzentriere mich im Folgenden auf zwei Fragestellungen: Wie lässt sich erklären, dass sich eine beträchtliche Anzahl von Entwicklern an Open Source Software Projekten beteiligt und wie wird die zu leistende Arbeit innerhalb eines Open Source Software Projektes unter den beteiligten Entwicklern koordiniert?

Den theoretischen Rahmen meiner Überlegungen bildet die Rational-Choice-Theorie (für einen Überblick siehe Diekmann & Voss 2004). Die RC-Theorie geht von einem Akteur aus, der sich an einer Entscheidungsregel orientiert, welche es ihm erlaubt, aus mehreren Handlungsalternativen jeweils die für ihn subjektiv vorteilhafteste zu wählen. Die RC-Theorie basiert somit auf der Analyse individueller Entscheidungen einzelner Akteure. Auch wenn die RC-Theorie von individuellen Akteuren ausgeht, ist sie jedoch primär an der Erklärung kollektiver Phänomene interessiert. Zudem werde ich meine Argumente primär auf die Spieltheorie als interaktionistisches Teilgebiet der RC-Theorie abstützen. Im Kontext der Entwicklung von Open Source Software geht es also darum, unter Berücksichtigung der individuellen Entscheidungen der einzelnen Projektmitglieder zu erklären, unter welchen Umständen Open Source Software produziert wird.[1]

Auf Entwickler, welche sich im Rahmen ihrer bezahlten Tätigkeit an einem Projekt beteiligen, ist die weitere Argumentation nur bedingt anwendbar. Die Entwickler sind

[1] Das hier präsentierte Modell ist nicht die einzige Möglichkeit, Open Source Entwicklung mit Konzepten der RC-Theorie zu erklären. Erwähnenswert sind in diesem Zusammenhang insbesondere die Arbeiten von Johnson (2002) und Bitzer & Schröder (2005). Johnson entwirft ein mikroökonomisches Modell des Entwicklungsprozesses. Er benutzt sein Modell jedoch insbesondere dazu, den Open Source Entwicklungsprozess mit der Entwicklung von proprietärer Software zu vergleichen. Daraus entwirft er zwar einige interessante Hypothesen zur Open Source Entwicklung, da sein Modell jedoch einen anderen Fokus hat, lassen sich seine Hypothesen nicht in meine Fragestellung integrieren. Bitzer & Schröder modellieren den Entwicklungsprozess mit Mitteln der Spieltheorie als «war of attrition». Sie beziehen in ihre Analyse zusätzlich eine zeitliche Komponente mit ein. Auch ihr Modell eignet sich nur bedingt, um daraus Hypothesen für meine Fragestellung abzuleiten. Es ist durch den Einbezug einer zeitlichen Komponente sehr komplex. Ich werde deshalb im Weiteren nicht näher auf diese beiden Modelle eingehen. Sie stehen jedoch auch nicht im Widerspruch zum hier vorgestellten Modell.

jedoch Akteure ihrer Arbeitgeber. Das hier entwickelte Modell kann ohne grössere Einschränkungen auf die Arbeitgeber übertragen werden.

3.1 Entwicklung von Open Source Software

Im Folgenden versuche ich darzulegen, wie die Entwicklung von Open Source Software mit Hilfe von zwei Spielen der Spieltheorie modelliert werden kann. Die Entwicklung von Open Source Software beinhaltet sowohl Elemente eines Gefangenendilemmas wie auch eines Freiwilligendilemmas. Wird der Entwicklungsprozess als ganzes betrachtet, kann er als iteriertes N-Personen Gefangenendilemma modelliert werden. Wird hingegen auf die Entwicklung einer bestimmten Programmfunktion oder die Behebung eines konkreten Fehlers fokussiert, dann gleicht dies einem Freiwilligendilemma. Es reicht, wenn ein Entwickler die Funktion programmiert oder den Fehler behebt damit alle davon profitieren können.

3.1.1 Open Source Software als kollektives Gut

Bei der Entwicklung von Open Source Software wird ein «kollektives Gut» [2] (Olson 1965) produziert. Ein kollektives Gut zeichnet sich durch die Absenz von Rivalität im Konsum und die Absenz von Ausschliessbarkeit aus (Varian 1992) [3] Der Konsum des Gutes durch eine Person beeinträchtigt den Konsum durch eine andere Person nicht. Dies ist eine der Grundeigenschaften aller digitalisierten Güter. Durch die unbeschränkte Kopierbarkeit dieser Güter herrscht keine Rivalität im Konsum. Diese Eigenschaft eines kollektiven Gutes trifft sowohl auf proprietäre, wie auch auf Open Source Software zu.[4] Da Open Source Software aber aufgrund ihrer speziellen Lizenzbedingungen ohne Gebühr an alle lizenziert wird, herrscht zusätzlich auch keine Ausschliessbarkeit beim Konsum. Diese Eigenschaft unterscheidet Open Source Software von proprietärer Software. Bei letzterer muss jeder Nutzer eine Lizenz für die Benutzung erwerben. Open Source Software erfüllt damit die Bedingungen eines reinen kollektiven Gutes.

Kollektive Güter bestehen oft aus einem gemeinsamen Gut, dessen Erhaltung durch die übermässige Nutzung durch die Konsumenten bedroht ist. Klassische Beispiele für diese Art eines kollektiven Gutes sind «saubere Luft» oder die «Allmende» (vergleiche Hardin 1968). Obwohl Open Source Software zuweilen auch als «digitale Allmende» (Grassmuck 2002) bezeichnet wird, unterscheidet sich die Problematik der Erstellung von Open Source Software grundlegend von dieser Art des kollektiven Gutes. Das Problem ist nicht die Übernutzung. Diese ist gar nicht möglich, da bei Open Source Software keine Rivalität

[2] In der Literatur wird sowohl der Begriff «kollektives Gut», als auch der Begriff «öffentliches Gut» verwendet. Da es sich bei Open Source Software um ein kollektiv erstelltes Gut und nicht um eine öffentliche Leistung handelt, werde ich den Begriff «kollektives Gut» verwenden.

[3] In der Literatur werden teilweise auch kollektive Güter diskutiert, welche nur eine dieser beiden Eigenschaften erfüllen. Insbesondere die Absenz von Rivalität trifft auf viele physische kollektive Güter nicht zu. Auf die Entwicklung von Open Source Software treffen jedoch beide Eigenschaften zu.

[4] Es ist in diesem Zusammenhang unerheblich, dass proprietäre Software oft durch Kopierschutzmassnahmen technisch gegen unkontrollierte Vervielfältigung gesichert ist. Entscheidend für die non-Rivalität im Konsum ist, dass der Inhaber des Copyrights die Software unendlich reproduzieren kann.

Abbildung 3.1: Auszahlungsmatrix Gefangenendilemma

	Kooperation	Defektion
Kooperation	r,r	s,t
Defektion	t,s	p,p

Anmerkung: Dabei gilt $t > r > p > s$, wobei t (Temptation) der Anreiz ist, einseitig zu defektieren, r (Reward), die Belohnung für wechselseitige Kooperation, p (Punishment), die Bestrafung für wechselseitige Defektion und s (Sucker's Payoff) die Auszahlung bei einseitiger Kooperation.

im Konsum herrscht. Vielmehr gilt es zu klären, unter welchen Umständen das kollektive Gut Open Source Software überhaupt produziert wird. Diese Art von kollektivem Gut entspricht viel mehr dem, was Olson (1965) beschreibt.

3.1.2 Gefangenendilemma

Spieltheoretisch wird die Produktion eines kollektiven Gutes oft als Gefangenendilemma modelliert (Hardin 1971). Im einfachsten Fall, dass nur zwei Personen zusammen ein kollektives Gut produzieren, entspricht die Produktion des Gutes dem klassischen 2-Personen Gefangenendilemma (Rapoport & Chammah 1965). Die Auszahlungsmatrix dieses Spiels ist in Abbildung 3.1 dargestellt.

Bei der Entwicklung von Open Source Software müssen sich alle Entwickler entscheiden, ob sie zum Projekt beitragen wollen, oder ob sie sich nicht an der Entwicklung beteiligen und nur vom Produkt profitieren möchten. Solange die Kosten der eigenen Beteiligung höher als der durch den eigenen Einsatz erzeugten Nutzen sind (d.h. $t > r$ im Gefangenendilemma), dann ist es für alle Beteiligten individuell rational, sich nicht zu beteiligen. Defektion ist im Gefangenendilemma eine dominante Strategie, da sie bei gegebener Wahl des anderen Entwicklers ein besseres Resultat liefert als Kooperation. Da die Auszahlungen im Gefangenendilemma symmetrisch sind, gilt dies jedoch für alle Beteiligten. Wechselseitige Defektion ist damit das einzige Gleichgewicht im Gefangenendilemma. Dieses Ergebnis der wechselseitigen Defektion ist jedoch für alle Beteiligten schlechter als das Ergebnis wechselseitiger Kooperation ($r > p$ im Gefangenendilemma). Deshalb wird das Gleichgewicht von Hamburger (1973) als mangelhaft («deficient equilibrium») bezeichnet. Das Grundproblem der Modellierung der Entwicklung von Open Source Software unter der Annahme von rationalen Entwicklern ist also, zu erklären, weshalb sich trotzdem eine genügende Anzahl von Entwicklern am Projekt beteiligt, obwohl dies auf den ersten Blick ihren rationalen Interessen zuwiderläuft.

Das bisher ausgeführte Modell des einmaligen Gefangenendilemmas kann jedoch die Entwicklung von Open Source Software nicht hinreichend erklären. Dazu sind zwei Erweiterungen notwendig. Erstens besteht die Entwicklung von Open Source Software in der Regel nicht aus einmaligen Interaktionen, sondern die Akteure beteiligen sich über eine längere Zeit an einem Projekt und müssen sich immer wieder entscheiden, ob und in welchem Umfang sie sich weiter engagieren möchten. Das Modell muss deshalb vom einmaligen zum iterierten Gefangenendilemma weiterentwickelt werden. Zweitens

sind an den meisten Open Source Projekten mehr als zwei Entwickler beteiligt. Das geschilderte Modell muss deshalb von einem 2-Personen auf ein N-Personen Spiel erweitert werden. Alternativ kann zumindest in gewissen Situationen die Auszahlungsstruktur nicht derjenigen des Gefangenendilemmas entsprechen. Ich werde auf diese Möglichkeit später noch detaillierter eingehen.

In iterierten Spielen treffen, im Gegensatz zu einmaligen Spielen, die gleichen Interaktionspartner mehrmals aufeinander. Solche Spiele werden auch «Superspiele» genannt. Um die Auszahlungen in iterierten Spielen bewerten zu können, muss jedoch berücksichtigt werden, dass zukünftige Auszahlungen in der Gegenwart einen geringeren Wert haben. Der Wert einer bestimmten Entwicklung, welche erst in der Zukunft verfügbar ist, ist entsprechend kleiner, da sie erst später genutzt werden kann. Auch kann das Verhalten des Partners in der Zukunft weniger genau vorhergesagt werden, da sich die relevanten Umstände verändern können. Deshalb werden die zukünftigen Auszahlungen mit einem Diskontfaktor w ($0 \leq w \leq 1$) bewertet. Mit dem Diskontfaktor kann auch berücksichtigt werden, dass das Spiel mit einer gewissen Wahrscheinlichkeit nach jedem Durchgang endet. Es besteht also eine gewisse Unsicherheit, ob die zukünftigen Auszahlungen überhaupt eintreffen werden.

Treffen Entwickler mehrmals aufeinander, dann können sie die Entscheidungen des Partners in den vorausgehenden Runden in ihrer eigenen Entscheidung berücksichtigen. Dadurch erhalten sie die Möglichkeit, über ihre Entscheidungen miteinander zu «kommunizieren». Sie können davon ausgehen, dass die anderen Beteiligten vergangene Resultate in ihren zukünftigen Entscheidungen auch berücksichtigen werden. Eine einseitige Defektion wird deshalb mit grösserer Wahrscheinlichkeit zu einer Defektion des Partners bei einem folgenden Aufeinandertreffen führen. Die Entwickler erhalten somit die Möglichkeit endogener Sanktionen. Das heisst, sie können nicht kooperatives Verhalten sanktionieren, ohne dass dazu eine übergeordnete Sanktionsmacht, beispielsweise der Staat, notwendig wäre. Dies ist insbesondere im freiwilligen Kontext der Entwicklung von Open Source Software von zentraler Bedeutung, da hier eine andere Sanktionsmacht vollständig fehlt.

Axelrod (1984) untersucht iterierte Gefangenendilemma analytisch und mit Hilfe einer Computersimulation verschiedener Strategien. Im Gegensatz zum einmaligen Gefangenendilemma gibt es im iterierten Spiel keine dominante Strategie. Welche Strategie am besten ist, hängt von der Strategie des Partners und der Bewertung zukünftiger Auszahlungen ab. Werden zukünftige Auszahlungen genügend hoch bewertet, dann können kooperative Strategien gegen gewisse Strategien des Partners bessere Ergebnisse liefern als vollständige Defektion. Dies gilt jedoch nur, falls das Spiel unendlich wiederholt wird. Andernfalls lässt sich durch Rückwärtsinduktion zeigen, dass sich rationale Akteure gleich wie im einmaligen Spiel verhalten werden (Luce & Raiffa 1957). In realen Situationen wie der Entwicklung von Open Source Software ist eine unendliche Iteration jedoch unmöglich. Jeder Entwickler wird früher oder später seine Beteiligung an einem Projekt beenden. Um das Argument der Rückwärtsinduktion zu entkräften reicht es jedoch, wenn die Entwickler nicht wissen, wann sie das letzte Mal aufeinander treffen werden. Ein erneutes aufeinandertreffen ist so lange wahrscheinlich, wie beide Entwickler im gleichen Projekt aktiv sind. Da der Ausstieg aus einem Projekt meist überhaupt nicht oder dann nicht im

voraus angekündigt wird, wissen sie nicht, wann sie das letzte Mal aufeinandertreffen.

Generell lassen sich Strategien nach zwei Eigenschaften charakterisieren. Erstens danach, ob sie im ersten Zug immer kooperieren oder nicht. Die kooperierenden Strategien werden von Axelrod «freundlich» genannt. Zweitens danach, ob eine Strategie «provozierbar» ist. Das heisst, ob sie auf eine Defektion des Partners in den folgende Zügen mit einer gewissen Wahrscheinlichkeit mit Defektion antwortet. In der Simulation hat sich gezeigt, dass alle freundlichen Strategien erfolgreicher als die unfreundlichen waren. Am erfolgreichsten war die die Strategie «Tit-for-Tat». Diese Strategie startet mit Kooperation und trifft danach immer die Wahl des Interaktionspartners in der vorhergehenden Runde. Diese Strategie kooperiert mit Partnern, welche auch kooperieren, lässt sich aber von nicht kooperierenden Partnern nur sehr bedingt ausnutzen. Sie ist sowohl freundlich, als auch schnell provozierbar.

Die Formeln 3.1 und 3.2 zeigen die Auszahlungen bei vollständiger Defektion («IMMERD») bzw. vollständiger Kooperation («IMMERK»), falls der Interaktionspartner die Strategie «Tit-for-Tat» wählt.

$$\begin{aligned} V(IMMERD|TFT) &= t + \sum_{i=1}^{\infty} w^i p = t + w \sum_{i=0}^{\infty} w^i p \\ &= t + w\frac{p}{1-w} \end{aligned} \tag{3.1}$$

$$\begin{aligned} V(IMMERK|TFT) &= \sum_{i=0}^{\infty} w^i r \\ &= \frac{r}{1-w} \end{aligned} \tag{3.2}$$

Daraus ergibt sich folgende Ungleichung für den Fall, dass die Auszahlung von «IMMERK» grösser als diejenige von «IMMERD» sein soll:

$$\begin{aligned} \frac{r}{1-w} &\geq t + \frac{wp}{1-w} \\ w &\geq \frac{t-r}{t-p} \end{aligned} \tag{3.3}$$

Die Ungleichung 3.3 zeigt, dass «IMMERK» bei genügend grossem Diskontfaktor w eine grössere Auszahlung erreicht als «IMMERD». «IMMERD» ist bei genügend grossem w keine dominante Strategie. Es gibt kooperative, aber provozierbare Strategien, wie «Tit-for-Tat» gegen die «IMMERD» nicht die beste Antwort ist. Allerdings ist auch «Tit-for-Tat» keine dominante Strategie.

Dass die «Tit-for-Tat» Strategie in der Open Source Entwicklung eine reale Rolle spielt, zeigt folgende Anekdote. Richard Stallman, der Gründer des GNU Projekts und der Free Software Fundation, weigert sich dem MIT «Lab for Computer Science» Zugang zu neuen Versionen seines «EMACS» Editors zu geben, da dieses seiner Meinung nach mit «faschistischen Sicherheitsmassnahmen» die Hacker Community am AI-Lab des MIT

«sabotierten». (Levy 1994: 293) Er wandte damit in einem gewissen Sinne die Tit-for-Tat Strategie an, in dem er nicht mit jener Abteilung kooperierte, die seiner Meinung nach zuvor nicht mit seiner Community kooperiert hatte. Eine Verallgemeinerung findet dieses Verhalten im Prinzip der Copyleft Lizenz.[5] Diese verhindert, dass Entwickler oder Unternehmen Open Source Programme in ihre eigenen Programme einbauen können, ohne ihre eigenen Weiterentwicklungen am Programm auch wieder zur Verfügung zu stellen. Damit wird eine für die Community besonders schädigende Form der Defektion, dass jemand die Software zwar weiterentwickelt, diese Verbesserung aber den anderen nicht zur Verfügung stellt, bereits durch die Lizenz verhindert. Copyleft erzwingt damit eine «Tit-for-Tat» Strategie. O'Mahony (2003) führt diese Argumentationslinie, dass sich Open Source Projekte durch rechtliche und institutionelle Vorkehrungen vor Ausbeutung schützen, weiter aus.

Das 2-Personen Gefangenendilemma kann zu einem N-Personen Spiel verallgemeinert werden. Das N-Personen Gefangenendilemma wurde insbesondere von Hardin (1971) als Modell für die Erstellung eines kollektiven Gutes vorgeschlagen. Hardin versuchte damit, die von Olson (1965) vorgelegte Untersuchung kollektiver Güter spieltheoretisch zu formalisieren.[6]

Im N-Personen Gefangenendilemma, wie es von Hamburger (1973) modelliert wird, spielt jeder Spieler simultan mit jedem anderen Spieler ein Gefangenendilemma, wobei ein Spieler in jedem dieser Teilspiele jeweils die gleiche Wahl (kooperieren oder defektieren) treffen muss. Er kann nicht mit einigen Spieler kooperieren und gegenüber anderen defektieren.[7] Die Auszahlungen hängen damit von der eigenen Wahl und der Anzahl der

[5] Open Source Software Lizenzen lassen sich grob in zwei Kategorien unterteilen. Lizenzen mit sog. Copyleft und solche, die Nutzung des Quellcodes in proprietären Anwendungen nicht einschränken. Das Prinzip des Copylefts besagt, dass bei der Weitergabe modifizierter Versionen des Open Source Programms immer auch der Quellcode der Modifikation mitgegeben werden muss. Das Copyleft schränkt die reine Verwendung der Software zusammen mit proprietärer Software nicht ein. Die wichtigste Copyleft Lizenz ist die GNU General Public License (GPL).

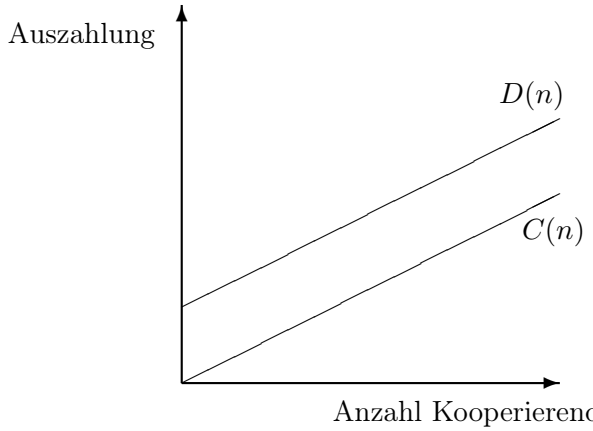
[6] Alternativ kann die Erstellung eines kollektiven Gutes spieltheoretisch auch mit dem «Public Good Game» modelliert werden. Beim Public Good Game erhalten alle Spieler einen bestimmten Betrag e und können entscheiden, welchen Anteil des Betrags c_i sie in einen gemeinsamen Topf legen wollen. Die Auszahlungsfunktion des Spieles wird dann wie folgt festgelegt:

$$\pi_i = e - c_i + f \cdot \frac{1}{N} \cdot \sum_{j=1}^{N} c_j \tag{3.4}$$

Wobei $1 < f < \frac{1}{N}$ ist. Die Spieler erhalten somit den zurückbehaltenen Anteil ihres ursprünglichen Betrags plus ihren Anteil am gemeinsamen Topf multipliziert mit einem Faktor. Der Gesamtnutzen aller Beteiligten ist am grössten, wenn alle Spieler den gesamten Betrag in das kollektive Gut investieren. Für jeden einzelnen Spieler ist es jedoch gewinnbringend, nicht in das kollektive Gut zu investieren, so lange die anderen das Gut weiterhin produzieren. Die Grundproblematik des «Public Good Game» ist gleich wie die des N-Personen Gefangenendilemmas. Das Spiel ist etwas differenzierter, da sich die Spieler nicht nur zwischen Kooperation und Defektion entscheiden können, sondern einen variablen Betrag beisteuern. Um die Analyse so einfach wie möglich zu halten, werde ich im Folgenden aber nicht weiter auf das «Public Good Game» eingehen.

[7] Hardin (1971) modelliert das Gefangenendilemma leicht anders. Sein Modell beschreibt jedoch eher ein kollektives Gut mit Rivalität im Konsum (d.h. Das Gut wird beim Konsum durch eine Person

Abbildung 3.2: Auszahlung im N-Personen Gefangenendilemma



Anmerkung: Die Funktionen $C(n)$ und $D(n)$ sind nur für $n \in \{0, 1, ..., N\}$ definiert. Ihre Darstellung als stetige Funktionen ist deshalb als näherungsweise Illustration zu verstehen.

anderen Spieler ab, die sich für Kooperation entscheiden. Sie können durch die Funktionen $C(n)$ respektive $D(n)$ dargestellt werden.

$$C(n) = nr + (N - n - 1)s = (r - s)n + s(N - 1) \tag{3.5}$$

$$D(n) = nt + (N - n - 1)p = (t - p)n + p(N - 1) \tag{3.6}$$

Wobei N die Anzahl der Spieler und n $(0 \leq n \leq N - 1)$ die Anzahl der anderen Spieler ist, welche kooperieren. Abbildung 3.2 zeigt $C(n)$ und $D(n)$ graphisch unter der Annahme, dass $r - s = t - p$ ist.

Damit hängt die Auszahlungsdifferenz zwischen Kooperation und Defektion nicht von der Wahl der Partner ab. Ein Wechsel von Kooperation zu Defektion bringt immer den gleichen absoluten Nutzenzuwachs. Übertragen auf die Entwicklung von Open Source Software bedeutet dies, dass der Nutzen der eigenen Entwicklung unabhängig vom Nutzen der Entwicklung des Partners ist. Die beiden Entwicklungen haben zusammen nicht einen grösseren Nutzen als der addierte Nutzen der Einzelentwicklungen. Diese Vereinfachung ist sinnvoll, wenn es sich bei den Beiträgen der beiden Entwickler um voneinander unabhängige Features der Software handelt, die je auch für sich nutzbar sind. In der Realität ist dies oft, aber nicht immer gegeben.

Dieses Spiel hat in den vier Extremfällen, dass alle Spieler defektieren, dass nur ein Spieler kooperiert, dass nur ein Spieler defektiert und dass alle Spieler kooperieren die gleiche Auszahlungsstruktur wie das 2-Personen Gefangenendilemma. Wie in Abbildung 3.2 ersichtlich ist gilt zudem für alle n dass $D(n) > C(n)$. Defektion ist damit, gleich wie im 2-Personen Spiel, auch im N-Personen Spiel eine dominante Strategie.

Falls $r - s > t - p$ und somit der Gesamtnutzen mit zunehmender Kooperation überproportional zunimmt, dann steigt $C(n)$ schneller als $D(n)$. Da in diesem Fall jedoch

verbraucht und steht danach anderen nicht mehr zur Verfügung). Deshalb wird sein Modell hier nicht näher betrachtet.

s kleiner beziehungsweise p grösser werden muss und die Auszahlungen unabhängig von der Anzahl Kooperationen von diesen Parametern abhängen, kann $C(n)$ nicht grösser als $D(n)$ werden. Der Anreiz zu defektieren nimmt jedoch in diesem Fall mit zunehmender Kooperation ab. $r - s > t - p$ gilt, falls der kumulative Nutzen einzelner Entwicklungen grösser als ihr addierter Einzelnutzen ist. Dies dürfte in Open Source Projekten vor allem zu Beginn der Entwicklung der Fall sein. In dieser Situation erodiert die Kooperation um so schneller, je weniger Entwickler zur Kooperation bereit sind. Dies könnte ein Grund sein, weshalb viele Open Source Projekte bereits in einem sehr jungen Stadium scheitern. Wie Koch (2005) zeigt, kommen viele Projekte auf der Open Source Plattform sourceforge.net nicht über das Anfangsstadium hinaus. Im Median haben die 8'621 untersuchten Projekte nur jeweils einen einzigen Entwickler und einen Entwicklungsstatus von 3 (Alpha Release) in einer Skala von 1 (Planning) bis 6 (Mature). Kontrastierend dazu sind jedoch die Erkenntnisse von Krishnamurthy (2002). Er hat nur die 100 aktivsten Projekte im Status «Mature» untersucht und herausgefunden, dass selbst unter diesen vergleichsweise erfolgreichen Projekten die meisten nur sehr wenige Entwickler haben. Eine Möglichkeit, wie sich dies erklären lässt, wird im Abschnitt 3.2.1 mit einer genaueren Betrachtung der Nutzenfunktion erläutert. Es braucht dazu Entwickler, für welche der Nutzen der eigenen Entwicklung grösser ist als die Kosten.

Analog zum iterierten 2-Personen Gefangenendilemma gibt es auch im iterierten N-Personen Spiel kooperative Gleichgewichte (Friedman 1971, Raub 1988). Allerdings sind die kooperativen Gleichgewichtsstrategien so beschaffen, dass sie bereits wenn ein einziger Spieler defektiert, in der nächsten Runde auch defektieren. Eine solche Strategie wird auch als «streng provozierbar» bezeichnet. Durch diese Strategien werden aber auch alle kooperierenden Spieler bestraft. Falls jedoch nicht bereits nach der ersten Defektion auch alle anderen Spieler auch defektieren, wäre es für jeden einzelnen Spieler individuell rational, als einziger zu defektieren und so ständig einen höheren Gewinn zu erzielen. In der Terminologie von Olson (1965) würden solche Spieler «Free Rider» genannt.

Raub (1988) argumentiert jedoch, dass es in N-Personen Dilemmata unter Umständen für eine Untergruppe vorteilhaft sein kann, weiter zu kooperieren, obwohl andere Spieler nicht kooperieren. Das Spiel kann dann als N-Personen Dilemma unter den anfänglich kooperierenden Personen betrachtet werden. Diejenigen, die von Beginn weg nicht kooperieren werden aus der Betrachtung ausgeschlossen. Diese Sichtweise erscheint insbesondere für die Situation der Entwicklung von Open Source Software angebracht, da es praktisch fast nicht möglich ist, dass sich alle Nutzer der Software auch an ihrer Entwicklung beteiligen. Ein grosser Pool von Nutzern der Software ist vielmehr Voraussetzung für eine überlebensfähige Community um ein Open Source Software Projekt. Aus diesem Pool von Nutzern lassen sich mit der Zeit neue Entwickler rekrutieren. Allerdings wird wohl nur ein kleiner Teil der Nutzer sich je aktiv an der Weiterentwicklung beteiligen. Mit einem grossen Nutzerkreis wächst jedoch auch die Chance, neue Entwickler zu gewinnen. Ich werde diesen Aspekt der «Entwicklerkarrieren» am Beispiel des Debian Projekts im Abschnitt 5.2 genauer analysieren. Das N-Personen Dilemma bezieht sich dann nur auf die aktiven Entwickler der Community und nicht auch auf die reinen Nutzer der Software. Allerdings ist Kooperation in dieser Situation nur im auf die Entwickler begrenzten Bereich eine Gleichgewichtsstrategie. Bezogen auf alle Akteure (Entwickler und Nutzer),

ist Kooperation jedoch keine Gleichgewichtsstrategie. Daraus ergibt sich die Frage, wie sich erklären lässt, dass gewisse Nutzer beginnen, sich aktiv an der Entwicklung eines Open Source Projekts zu beteiligen, währenddem der grösste Teil die nach dem bisherigen Erläuterungen rationale Strategie des «free-riding» wählen und die Software nur Nutzen, ohne sich aktiv zu beteiligen.

Iterierte N-Personen Gefangenendilemmata wurden durch Diekmann & Manhart (1989), in einem an Axelrod (1984) angelehnten Verfahren, simuliert. Die Simulationen umfassten 13 Strategien, 8 davon sind freundlich. Die Simulation wurde sowohl mit streng provozierbaren Varianten der Strategien, welche bereits die Defektion einer Strategie als Defektion werten, wie auch mit nachsichtigeren Varianten, welche erst bei 50% Defektion reagieren, durchgeführt. Die Simulationen wurden mit unterschiedlichen Gruppengrössen zwischen 2 und 30 Strategien durchgeführt. Die Gruppengrösse hatte denn auch einen entscheidenden Einfluss auf den Erfolg einer Strategie. Währenddem in kleinen Gruppen gleich wie in Axelrod's Simulation die freundlichen Strategien durchwegs besser abschnitten, wurden die unfreundlichen Strategien mit zunehmender Gruppengrösse erfolgreicher. Wobei die streng provozierbaren Strategien weniger stark von den unfreundlichen Strategien ausgenutzt wurden, als die nachsichtigeren. In kleinen Gruppen mit bis zu 6 Strategien waren jedoch die nachsichtigeren besser. In Gruppen mit $N \geq 15$ schnitt jeweils die Strategie «IMMER D» am besten ab. Im Durchschnitt schnitten jedoch in allen Gruppen die freundlichen Strategien besser ab als die unfreundlichen. In grösseren Gruppen war der Vorsprung der freundlichen Strategien jedoch deutlich kleiner. Mit steigender Gruppengrösse verringerten sich auch die durchschnittlichen Auszahlungen. Die lässt sich durch die starke Abnahme kooperativer Entscheidungen erklären. So sank die Kooperationsrate von 81% bei $N = 2$ auf 19% bei $N = 30$.

Dass Kooperation in iterierten N-Personen Gefangenendilemmata zwar analytisch begründbar, aber äusserst fragil ist, lässt sich durch ein Argument von Diekmann (1993*b*) zeigen. Dieses geht davon aus, dass die Spieler mit einer kleinen Wahrscheinlichkeit p eine Fehler machen und nicht die Wahl treffen, die eigentlich ihrer Strategie entspricht oder die Wahl eines Mitspielers falsch interpretieren. Formel (3.7) zeigt, demzufolge die Wahrscheinlichkeit $p*_R$, dass bis zur Runde R kein Fehler passiert, mit zunehmender Spielerzahl N und mit zunehmender Zahl der Iterationen R rasch abnimmt.

$$p^* = 1 - \underbrace{(1-p)}_{keinFehler}{}^{NR} \tag{3.7}$$

Da jedoch nur streng provozierbare Strategien Gleichgewichtsstrategien sein können, wird die Kooperation durch einen einzigen Fehler zerstört. Mit der Wahrscheinlichkeit p^* endet sie nach der Runde R. Bei einer Fehlerwarscheinlichkeit von 1% ($p = 0.01$) und 10 Spielern beträgt p^* bereits 0.63.

3.1.3 Freiwilligendilemma

Die vorhergehende Betrachtung der Entwicklung als Gefangenendilemma fokussierte auf den gesamten Open Source Entwicklungsprozess. Wird hingegen die Entwicklung eines bestimmten Features oder die Behebung eines bestimmten Fehlers einzeln betrachtet, so

Abbildung 3.3: Auszahlungsmatrix Freiwilligendilemma (Diekmann 1985)

Anzahl C Wahlen	0	1	2	...	N
C (Gut erstellen)	—	$U - K$	$U - K$	...	$U - K$
D (Gut nicht erstellen)	0	U	U	...	—

Anmerkung: $N \geq 2$; $U - K > 0$

ergibt sich ein etwas anderes Bild. Die Entwicklung gleicht dann eher dem von Diekmann (1985) beschriebenen N-Personen Freiwilligendilemma. In diesem Spiel müssen die Akteure zusammen ein kollektives Gut herstellen. Jeder einzelne Akteur kann entscheiden, ob er das Gut selbst herstellen will, oder ob er untätig bleibt. Das Gut hat den Nutzen U und bei der Erstellung entstehen Kosten von K. Der Nutzen des Gutes ist unabhängig von der Anzahl der Personen, welche das Gut produzieren. Im Idealfall produziert also nur jeweils ein Akteur das gut für Alle. Falls beispielsweise der gleiche Fehler von mehr als einer Person behoben wird, oder das gleiche Feature mehrmals implementiert wird, entsteht kein zusätzlicher Nutzen. Die Auszahlungen im Freiwilligendilemma sind in Abbildung 3.3 dargestellt.

Im Gegensatz zum Gefangenendilemma wird beim Freiwilligendilemma normalerweise davon ausgegangen, dass $U > K$ ist. Das heisst, dass für einen Entwickler alleine aus seiner eigenen Entwicklung ein grösserer Nutzen als Kosten entstehen. Falls die Kosten grösser als der Nutzen sind, dann ist Defektion eine dominante Strategie. Kooperation ist dann nur im iterierten Fall rational begründbar. In dieser Situation gleichen jedoch jeweils zwei Runden einem sequentiellen N-Personen Gefangenendilemma. Deshalb werde ich auf diesen Fall nicht weiter eingehen.

Im Freiwilligendilemma gibt es keine dominante Strategie. Die kollektiven Auszahlungen sind am höchsten, wenn nur ein Spieler das Gut herstellt. Für dieses optimale asymmetrische Gleichgewicht, in dem in jeder Runde nur ein Akteur das Gut produziert, müssen sich die Akteure jedoch koordinieren. Dies führt zum Dilemma, wie der Akteur bestimmt wird, der die Kosten auf sich zu nehmen hat und wie verhindert werden kann, dass jeder Akteur davon ausgeht, dass ein Anderer das Gut produzieren wird. So wird das Gut eventuell überhaupt nicht produziert, obwohl dies für keinen der Akteure eine dominante Strategie ist. Zieht man auch gemischte Strategien in Betracht, dann existiert eine weitere Gleichgewichtsstrategie. Die erwartete Auszahlung dieser Strategie ist jedoch $U - K$ und damit gleich wie bei einer C-Wahl. Dieses Gleichgewicht ist somit nicht profitabel.

Für beide Lösungen ist zu erwarten, dass die Wahrscheinlichkeit, dass mindestens ein Akteur das Gut herstellt, mit der Gruppengrösse abnimmt. Für die gemischte Strategie zeigt dies Diekmann (1993*a*) sowohl analytisch wie auch experimentell. Für das asymmetrische Gleichgewicht wird der Koordinationsaufwand mit zunehmender Gruppengrösse zunehmen. Damit nimmt auch die Chance eines Koordinationsfehlers zu. In der Sozialpsychologie ist dieser Effekt auch unter dem Stichwort «Verantwortungsdiffusion» bekannt.

3.2 Auswege aus dem Dilemma

Das im vorhergehenden Abschnitt entworfene Modell der Open Source Software Entwicklung wirft einige Fragen auf, weshalb unter diesen Bedingungen überhaupt Open Source Entwicklung stattfindet. In diesem Abschnitt werden Auswege aus diesen Dilemmata skizziert. Dabei werde ich von der streng spieltheoretischen Argumentation teilweise abweichen und auch alternative Sichtweisen präsentieren.

Dass Kooperation unter rationalen Akteuren im Gefangenendilemma prinzipiell möglich ist, wurde bereits im vorhergehenden Abschnitt gezeigt. Insbesondere die Simulationsexperimente von Diekmann & Manhart (1989) haben aber gezeigt, dass Kooperation mit steigender Gruppengrösse sehr fragil wird. In vielen Open Source Projekten ist die relevante Gruppe jedoch grösser als die maximal simulierte Gruppengrösse von 30. Das Debian Projekt enthält je nach Zeitpunkt bis zu 700 gleichzeitig aktive Entwickler. In einer solch grossen Gruppe kann Kooperation nicht mehr stabil sein, wenn bereits nach der Defektion eines einzigen Gruppenmitglieds alle anderen ebenfalls defektieren.

Bereits Olson (1965) hat in diesem Zusammenhang die zentrale Bedeutung selektiver Anreize erkannt. Solche selektiven Anreize können bei der Entwicklung von Open Source Software unterschiedlichste Formen annehmen. Sie führen dazu, dass sich die Kosten der Entwicklung für einige Spieler verringern. Für diese ändert sich die Auszahlungsmatrix so, dass der Unterschied zwischen dem «Suckers Payoff» s und dem «Punishment» p kleiner wird. Im Idealfall wird sogar $s > p$, so dass sich der Entwicklungsaufwand für diesen Entwickler unabhängig vom Verhalten der anderen einen positiven Nutzen hat. Für einen solchen Entwickler verwandelt sich das Gefangenendilemma in ein Chicken Spiel.[8] Von Hippel & von Krogh (2003) argumentieren, dass Open Source Entwicklung stark auf selektiven Anreizen beruht. In ihrem «Private-Collective Innovation» Modell versuchen sie Elemente privater Investition und kollektiver Güter miteinander zu verbinden. In ihrem Modell profitieren die Entwickler beispielsweise von der schnelleren Diffusion einer Open Source Entwicklung und der damit verbundenen grösseren Verbreitung der Software und dem Feedback zu ihrer Entwicklung. Free-Rider sind dann nicht mehr Profiteure, die nichts beitragen, sondern sie vergrössern den Marktanteil und die Wichtigkeit des Programms. Sie haben also auch einen Wert für das Open Source Projekt.

Wie bereits weiter oben ausgeführt, kann es im N-Personen Gefangenendilemma auch für Untergruppen profitabel sein, zu kooperieren, obwohl Andere sich nicht beteiligen. In den bisherigen Erörterungen wurde ein Open Source Projekt jeweils als eine monolithische Einheit betrachtet. In den meisten grösseren Projekten gibt es jedoch Subgruppen, die sich um Teilbereiche der Software kümmern. Im Debian Projekt ist die Bildung solcher Subgruppen sogar sehr ausgeprägt. Oft kümmern sich ein einziger oder einige wenige Entwickler um ein Paket oder ein Teilprojekt. In diesen kleinen Gruppen ist Kooperation bereits weit wahrscheinlicher. Im einfachsten Fall eines einzigen Zuständigen reduziert sich ein Fehlerbericht sogar auf das unten beschriebene Vertrauensspiel.

[8] Das Chicken Spiel wird auf Deutsch auch Feiglings-Spiel genannt. Es unterscheidet sich vom Gefangenendilemma dadurch, dass Defektion keine dominante Strategie ist. Jedoch ist auch Kooperation gleich wie im Gefangenendilemma keine dominante Strategie. Das Spiel hat in reinen Strategien zwei asymmetrische Nash-Gleichgewichte.

Wird ein Open Source Projekt in mehrere Teilprojekte unterteilt, dann eröffnen sich einem Entwickler auch neue Möglichkeiten der selektiven Kooperation. Er muss dann nicht mehr notwendigerweise entweder mit allen oder mit niemandem kooperieren, sondern kann in denjenigen Teilprojekten kooperieren, in denen ein grosser Anteil der Mitentwickler auch kooperiert. So kann das Problem, dass in grossen Gruppen Kooperation sehr instabil ist, entschärft werden. Auch wird durch den Zusammenbruch der Kooperation in einem Teilprojekt nicht unmittelbar die Kooperation im ganzen Projekt zerstört.

Eine mögliche Form eines solchen Teilprojekts, auf die ich im empirischen Teil der Untersuchung näher eingehen werde, ist das beheben eines Fehlers. Ein Fehlerbericht kann spieltheoretisch als sequentielles Gefangenendilemma betrachtet werden, in dem zuerst der Fehlerberichterstatter vor der Entscheidung steht, ob er einen Fehler überhaupt berichten soll und danach der zuständige Entwickler entscheiden muss, ob er den Fehler beheben will. Ein solches Spiel wird auch Vertrauensspiel genannt. Zentral für die Entscheidung, ob ein Entwickler kooperiert oder nicht, sind Informationen zur Kooperationswahrscheinlichkeit seines Gegenübers. Im empirischen Teil werde ich jedoch nur die zweite Entscheidung, also ob ein Fehler behoben wird oder nicht, betrachten.

Im Freiwilligendilemma präsentiert sich das Problem etwas anders. Hier geht es primär darum, wie die Gruppe ihre Auszahlungen maximieren kann, ohne dass dadurch die Produktion des kollektiven Gutes gefährdet wird. Wie können sich die Entwickler so koordinieren, dass jeweils nur einer an einem bestimmten Feature oder Fehler arbeitet. Wie wird der Freiwillige bestimmt, der jeweils die Kosten auf sich zu nehmen hat. Interessant ist dabei insbesondere, ob die Koordinationsfähigkeit von der Gruppengrösse abhängig ist.

Zur Lösung dieser Probleme werde ich zuerst die möglichen selektiven Anreize anhand der Kosten- und Nutzenfunktion von Open Source Entwicklern genauer unter die Lupe nehmen. Danach werde ich einen dieser Anreize, den Reputationsgewinn, genauer betrachten und zu erklären versuchen, wie Reputation zur Stabilisierung der Kooperation beitragen kann.

3.2.1 Kosten- und Nutzenfunktion

Der Nutzen einer Open Source Entwicklung beschränkt sich nicht auf deren direkten Nutzen für den Entwickler. Um eine vollständige Nutzenfunktion zu erhalten, müssen weitere Faktoren miteinbezogen werden, insbesondere der indirekte finanzielle Gewinn, der Gewinn an Reputation und der intrinsische Nutzengewinn während der Entwicklung.

$$U = f(U_{dev}, U_{fin}, U_{repint}, U_{repext}, U_{int}) \tag{3.8}$$

U_{dev} Direkter Nutzen der Entwicklung für den Entwickler, falls dieser die neue Funktion selbst benötigt.

U_{fin} Indirekter finanzieller Nutzen der Entwicklung durch den Verkauf von Dienstleistungen rund um die Entwicklung. Auch wenn für Open Source Software keine Lizenzgebühren verlangt werden können, kann mit einem geeigneten Geschäftsmodell durchaus Geld mit einem Open Source Produkt verdient werden. Beispielsweise

mit Dienstleistungen zur Software. Die Entwickler der Software sind sicherlich in der besten Position, solche Dienstleistungen anzubieten.

U_{repint} Nutzen durch höhere Reputation innerhalb des Open Source Software Projekts. Insbesondere durch grössere Sichtbarkeit, Bekanntheit und Vertrauenswürdigkeit.

U_{repext} Nutzen der höheren Reputation ausserhalb der Open Source Community. Insbesondere durch bessere Chancen am Arbeitsmarkt, höheres Gehalt oder zum Aufbau eines eigenen Unternehmens.

U_{int} Intrinsischer Nutzen durch Freude am Programmieren, Stolz auf die geleistete Arbeit und den dabei erzielten Lerneffekt.

Diese Komponenten der Nutzenfunktion ergeben sich auch aus den von Grassmuck (2002: 249-254) angeführten Motivationsfaktoren für die Entwicklung von Open Source Software. Auch die in Abschnitt 2.1.1 zitierten Studien kommen zu ähnlichen Ergebnissen. Lakhani & Wolf (2005) haben die Motivationsfaktoren detaillierter untersucht. Sie nehmen eine etwas andere Unterteilung vor, kommen aber nicht zu anderen Schlüssen. Viele dieser Komponenten werden auch von Raymond (2001) erwähnt, der die Open Source Entwicklung aus einer Innenperspektive analysiert. Im folgenden Zitat über die Nutzenfunktionen eines «Linux Hackers» benennt er explizit den intrinsischen Nutzen und die interne Reputation.

> «The "utility function" Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers.» (Raymond 2001: 53)

Hars & Ou (2002) haben die Motivation, sich an der Open Source Entwicklung zu beteiligen, mit einer Umfrage untersucht. Sie teilen die Motivationsfaktoren primär in zwei Kategorien: interne Faktoren und externe Belohnungen. Zu den internen Faktoren zählt insbesondere der intrinsische Nutzen (U_{int}). Bis zu einem gewissen Grad können auch Teile der Reputation innerhalb des Projekts (U_{repint}) als interne Faktoren angesehen werden, da sie zu diesen Faktoren auch die Identifikation mit der Community zählen. Eine grössere Reputation innerhalb des Projekts stärkt wohl auch die Position in und Identifikation mit der Community. Zu den externen Belohnungen gehören der direkte Nutzen der Entwicklung (U_{dev}), der indirekte finanzielle Nutzen (U_{fin}) und die gewonnene Reputation (U_{repext} und Teile von U_{repint}). Hars & Ou (2002) haben zudem herausgefunden, dass Studenten und Hobbyprogrammier die internen Faktoren höher gewichten, währendem angestellte und freischaffende Programmierer mehr Wert auf die externen Belohnungen legen.[9]

Dass der Aufbau einer externen Reputation durchaus einen sehr realen Einfluss auf zukünftige Chancen auf dem Arbeitsmarkt hat, zeigt folgendes Zitat aus einem Bericht über eine auf Open Source Entwickler spezialisierten Rekrutierungsfirma:

[9] Für eine weitere empirische Untersuchung zu den Motiven von 141 Linux Kernel Entwicklern, welche im Grossen und Ganzen zu ähnlichen Ergebnissen kommt, siehe auch Hertel et al. (2003).

> «If you see there's 10 or 20 IBM engineers doing this, if you get involved and contribute code, I would be willing to bet that at some point IBM is going to more than likely offer you a position.»[10]

Einige der Nutzenkomponenten kommen auch Free-Ridern zu Gute. Auch sie können von direkten Nutzen der Entwicklung (U_{dev}) und vom finanzielle Nutzen (U_{fin}) profitieren. Allerdings dürften beide Komponenten für sie im allgemeinen Schwächer sein, da sie keinen Einfluss auf die genaue Ausgestaltung der Entwicklung haben und weil sie zwar auch Zusatzdienstleistungen anbieten können, dazu aber in einer schlechteren Position sind. Der Nutzen der Entwicklung ist also für die Free-Rider geringer als für den Entwickler.

Allerdings hat der Entwickler alleine die Kosten der Entwicklung zu tragen. Die Kostenfunktion setzt sich aus den Opportunitätskosten C_{opp} und den direkten Kosten C_{dir} zusammen.

$$C = f(C_{opp}, C_{dir}) \tag{3.9}$$

Auf der Kostenseite fallen vor allem die Opportunitätskosten ins Gewicht. Durch die freiwillige Arbeit an einem Open Source Projekt verzichtet ein Entwickler auf das Einkommen, welches er in dieser Zeit als professioneller Entwickler erzielen könnte. Diese Kosten dürften in der Regel relativ hoch sein, da Softwareentwickler gut bezahlt sind. Auch dürften sich die Entwickler von Open Source Software meist durch ein überdurchschnittliches Können auszeichnen. Diese Kosten steigen mit dem Alter des Entwicklers. Als junger Schüler oder Student hat er wesentlich geringere Möglichkeiten, eine gute bezahlte Anstellung zu erhalten, als nach Abschluss eines Hochschulstudiums. Auch mit der zunehmenden Erfahrung in der Softwareentwicklung steigen die Opportunitätskosten. Für Entwickler, welche neben einer bezahlten Arbeit in der IT-Industrie in ihrer Freizeit Open Source Software entwickeln, fallen weniger Opportunitätskosten durch den Wegfall von Einkommen an als immaterielle Kosten durch den Verzicht auf eine andere Tätigkeit in der Freizeit.

Für selbständige Entwickler fällt weniger ein Einkommensverzicht ins Gewicht, als die ausfallenden Einnahmen aus der Lizenzierung, die bei einer proprietären Entwicklung realisiert werden könnten. Allerdings fallen auch bei der proprietären Entwicklung der Software Opportunitätskosten durch den Verzicht auf die Vorteile der Open Source Entwicklung an.

Im Vergleich zu den Opportunitätskosten sind die direkten Kosten relativ gering. Zur Beteiligung an der Entwicklung ist nur ein Computer und ein Internetanschluss nötig. Beides ist wahrscheinlich bei den meisten Entwicklern bereits vorhanden.

Insbesondere die Betrachtungen zur Nutzenfunktion zeigen, dass im Open Source Entwicklungsprozess zahlreiche Möglichkeiten für selektive Anreize vorhanden sind. Zudem sind auch die Kosten der einzelnen Entwickler je nach ihrer persönlichen Situation sehr unterschiedlich. Dies eröffnet zahlreiche Möglichkeiten, das individuelle Engagement eines

[10] Brent Marinaccio, Hot Linux Jobs, `http://itmanagement.earthweb.com/career/article.php/3774811`, Stand 2.1.2009

Entwicklers durch eine besonders starke Ausprägung einer der Komponenten der Nutzenfunktion oder durch besonders tiefe Opportunitätskosten zu erklären. Im folgenden Abschnitt möchte ich auf die Reputation als eine dieser Komponenten vertieft eingehen. Die anderen Komponenten werden im empirischen Teil der Arbeit weitgehend in den Hintergrund rücken. Dies hat aber weniger damit zu tun, dass sie eine geringere Erklärungskraft besitzen, als damit, dass sie sich mit dem vorliegenden Datensatz nicht beobachten lassen.

3.2.2 Reputation als Koordinationsmechanismus

In der allgemeinsten Definition versteht man unter Reputation meistens eine Eigenschaft, welche ein oder mehrere Akteure einem anderen zuschreiben (Wilson 1985). Damit verbunden wird oft auch eine Aussage über das wahrscheinliche zukünftige Verhalten des Gegenübers. Diese Betrachtung von Reputation als Wahrscheinlichkeit eines bestimmten Verhaltens nimmt Dasgupta (1988) für seine Definition der Reputation auf:

> «A reputation for honesty, or thrustworthiness, is usually acquired gradually. This alone suggests that the language of probabilities is the right one in which to discuss reputation: a person's reputation is the 'public's' imputation of a probability distribution over the various types of person that the person in question can be in principle.» (Dasgupta 1988: 62)

Mit dieser Definition ist die Reputation eines Entwicklers die Wahrscheinlichkeit, mit der er sich am Projekt beteiligen wird. Dabei wird davon ausgegangen, dass es zwei Typen von Entwicklern gibt: Solche, die grundsätzlich kooperieren und Free-Rider, welche nur von der Arbeit der anderen profitieren möchten. Die Reputation ist dann eine Wahrscheinlichkeitsaussage darüber, welchem Typ eine Person entspricht. Eine positive Reputation wird zu einem Signal an die anderen Entwickler. Diese können damit rechnen, dass Alter mit einer hohen Wahrscheinlichkeit kooperieren wird, was wiederum für sie bedeutet, dass die Chance sinkt, bei Kooperation ausgebeutet zu werden. Falls alle Akteure davon ausgehen, dass Reputation bei der Entscheidung über Kooperation eine zentrale Rolle spielt, dann lohnt es sich auch für die sonst nicht kooperationswilligen, zu kooperieren. Es ist dabei nicht wichtig, dass sich die gleichen Partner nochmals treffen. Es reicht anzunehmen, dass die Information über die Unzuverlässigkeit des Defektors allen anderen bekannt ist. Die Kooperation lohnt sich, da der Defektor andernfalls davon ausgehen muss, dass im weiteren Verlauf nicht mehr mit ihm kooperiert wird, was für ihn bei genügend hohem Wert der Zukunft eine schlechtere Auszahlung bedeutet. Reputation ermöglicht somit eine indirekte «Tit-for-Tat» Strategie.

Granovetter (1985) postuliert, dass der Einbettung der Akteure in soziale Beziehungen (Embededness) bei der Erklärung von kooperativem Verhalten zentrale Bedeutung zukommt. Kooperation wird nach ihm nicht primär durch soziale Normen, sondern durch Netzwerke sozialer Beziehungen zwischen den Akteuren hergestellt. Doch gerade diese soziale Einbettung ist in Open Source Communities auf den ersten Blick sehr schwach. Viele der beteiligten Personen haben nur sehr beschränkte Informationen übereinander

und interagieren ausschliesslich über elektronische Kommunikation und im Kontext des Projekts miteinander. In Open Source Projekten fehlt die gemeinsame raum-zeitliche Anwesenheit meist. Granovetter argumentiert, dass in solchen Situationen generalisierte Information eine grosse Bedeutung erhält, da keine besseren Indikatoren vorhanden sind. Wenn immer möglich versuchen wir jedoch, das Verhalten anderer aus eigener Erfahrung oder wenigstens aus Informationen vertrauenswürdiger Dritter abzuleiten, was meist bedingt, dass wir entweder bereits mit ihnen interagiert haben oder zumindest vertrauenswürdige Informationen über eine solche Interaktion bekommen können. Im Gegensatz zu Open Source Projekten haben die Akteure in vielen klassischen sozialen Situationen weit weniger Informationen über das vergangene Verhalten der Anderen, da sie dieses nicht direkt beobachten können. Reputation besteht dann vor allem aus Hörensagen. Dies erklärt Granovetters Skepsis gegenüber Reputation als Koordinationsmechanismus.

Die im Kontext der Open Source Entwicklung zentrale Komponente der Reputation ist die Reputation als kooperationsbereiter Entwickler. Diese lässt sich aus vergangener Kooperation ablesen. Die Information über vergangene Kooperation ist jedoch nicht nur den direkt betroffenen zugänglich. Dadurch, dass der grösste Teil der Entwicklung öffentlich stattfindet, haben auch alle anderen Entwickler Zugang zu dieser Information. Reputation ergibt sich somit aus dem vergangenen Einsatz für das Projekt. Dieser ist ein Signal an die anderen Entwickler zur Bereitschaft, auch in Zukunft zum Projekt beizutragen. Mit dieser Form der Reputation kann Kooperationsbereitschaft jedoch nur positiv vermittelt werden. Es gibt keine Indikatoren für Personen, die nicht zur Kooperation bereit sind, da sich dies nur dadurch ausdrückt, dass sie die Software nur nutzen, ohne etwas zur Entwicklung beizutragen. Dies erschwert die Diskriminierung von Free-Ridern, da nicht ersichtlich ist, ob eine Person erst gerade zum Projekt gestossen ist und grundsätzlich kooperationswillig ist oder ob sie bereits seit längerem defektiert. Andererseits ist es wahrscheinlich, dass die meisten Entwickler vor ihrer aktiven Beteiligung an einem Projekt dieses über eine gewisse Zeit passiv nutzen. Es ist deshalb für das Überleben des Projekts wichtig, dass es auch Free-Rider, welche nun aktiv beitragen wollen aufnehmen kann.

Dass der Einsatz für das Projekt in der Vergangenheit einen wichtigen Einfluss auf den Status eines Entwicklers hat, wird auch durch folgende Aussage eines der aktivsten Debian Entwickler illustriert. Es handelt sich dabei um einen Kommentar zu einer Person auf einer zentralen Mailingliste, welche mehrheitlich die Entscheide des Projekts kritisierte ohne aktiv zur Verbesserung beizutragen.

> «[...] statements from people who *do* things always carry more weight with me than people whose primary activity is making statements.»[11]

In dieser Aussage kommt sowohl zum Ausdruck, dass Reputation über das aktive Engagement innerhalb des Projekts entsteht, wie auch, dass die so gewonnene Reputation bei der Bewertung von Beiträgen anderer Entwickler eine zentrale Rolle spielt. Weiter wird auch deutlich, dass nicht jede Form der Mitarbeit den gleichen Stellenwert besitzt. Reine Beteiligung auf der Mailingliste hat einen geringeren Wert und kann, insbesondere wenn

[11] Steve Langasek am 22. September 2008 auf der debian-vote Mailingliste. `http://lists.debian.org/debian-vote/2008/09/msg00022.html`, Stand 26.12.2008

es sich vornehmlich um Kritik handelt, als eine Form der Defektion betrachtet werden, die zu einer negativen Reputation führt. Einen weiteren Hinweis auf die Bedeutung von Reputation liefert ein Ergebnis aus der bereits erwähnten FLOSS-POLS Studie. 94% der Entwickler markieren ihre Beiträge zu einem Projekt und 58% geben an, dass diese sehr wichtig sei (Ghosh et al. 2002: 62f.).

Wehrli (2005) hat mit seiner Untersuchung über Reputationseffekte bei Online-Auktionen in einem etwas anderen Kontext gezeigt, dass Reputation in Online Communities eine zentrale Rolle spielen kann. Ähnlich wie bei der Entwicklung von Open Source Software interagieren in Online-Auktionen anonyme Fremde miteinander, ohne dass sie über die in klassischen Situationen üblicherweise verfügbaren Absicherungsmechanismen und Sanktionsmöglichkeiten verfügen. In Face-to-Face Situationen begünstigt die weit geringere Anonymität Kooperation. Die Analyse von eBay hat gezeigt, dass die Reputation einen positiven Einfluss auf den erzielten Verkaufspreis hat.

Anders als im Debian Projekt gibt es in der von Wehrli (2005) untersuchten Plattform eBay einen eingebauten Mechanismus zur Bewertung früherer Interaktionen, der für alle Partner zugänglich ist. Zwar sind praktisch alle Vorgänge innerhalb des Debian Projekts öffentlich dokumentiert. Will sich jedoch ein Entwickler ein Bild über das Verhalten eines Anderen machen, muss er sich diese Informationen erst aus verschiedenen Quellen zusammensuchen, was mit nicht unerheblichen Kosten verbunden ist. Es ist deshalb anzunehmen, dass ein Entwickler jeweils nur über unvollständige Informationen verfügt und dies Informationen wesentlich genauer und detaillierte sind, falls die Akteure in früheren Situationen bereits miteinander interagiert haben. Die Verbreitung der Reputation dürfte deshalb in der Realität einer Netzwerkstruktur folgen. Zentrale Knoten in diesem Netzwerk sind diejenigen Entwickler, die sehr viel Arbeit leisten. Ihre Reputation ist deshalb sehr vielen anderen Entwicklern bekannt. Auch verfügen sie über Informationen über einen grösseren Anteil der anderen Entwickler.

Zusätzlich zur geleisteten Arbeit gibt es im Debian Projekt den offiziellen Entwicklerstatus als einen weiteren Reputationsindikator. Dieser ist oft sehr einfach ersichtlich, da die meisten Entwickler ihre Debian Mailadresse für ihre Projektarbeit benutzen. Auch ist die Liste aller offiziellen Entwickler einfach zugänglich. Der Entwicklerstatus ist jedoch alles andere als unabhängig von der geleisteten Arbeit. Damit jemand offizieller Entwickler wird, muss er bereits einen substantiellen Beitrag geleistet haben. Der Entwicklerstatus ist damit vielmehr ein vereinfachender Proxy für die geleistete Arbeit und damit für eine positive Kooperationsreputation.

Die durch die schwache soziale Einbettung reduzierte Unmittelbarkeit der Reputationssignale wird somit dadurch wettgemacht, dass die generalisierten Reputationssignale nicht aus möglicherweise verfälscht wiedergegebener Erfahrung anderer besteht, sondern aus direkt beobachtbaren und damit unverfälschten Handlungen. Granovetters Argument, dass generalisierte Informationen einen geringen Wert besitzen, wird damit zumindest abgeschwächt.

Mui, Mohtashemi & Verma (2004) wenden das Freiwilligendilemma auf die Entwicklung von Open Source Software an. Sie entwickeln ein Modell, in welchem sie zeigen, wie die Akteure zu Informationen über das wahrscheinliche Verhalten Alters kommen. Direkte Reziprozität (d.h. dass Alter im Gegenzug eine für Ego direkt nutzbare Entwicklung

beisteuert), scheint bei grösseren Open Source Projekten eher unwahrscheinlich. Die Entwickler müssen sich bei ihrem Verhalten also auf eine indirekte Form von Reziprozität stützen. Mui et al. vermuten daher, dass Open Source Entwickler durch den damit verbundenen Reputationsgewinn zur Kooperation motiviert werden. Diese Vermutung wird auch durch zahlreiche andere Untersuchungen zu Open Source Software geäussert (beispielsweise Raymond 2001, von Hippel & von Krogh 2003, Brand n.d.*b*). Open Source Projekte werden von diesen oft als Meritokratie beschrieben, in welcher Status und Reputation über den Wert der für das Projekt geleisteten Arbeit zugeschrieben wird.

> «Der innere Kreis ist eine schwer abgrenzbare Gruppe von Personen, die sich um das Projekt verdient gemacht und deswegen Reputation bzw. sozialen Status erlangt haben. Die Reputation besteht aus verschiedenen Elementen, wie Seniorität/ Erfahrung, kontinuierliche Leistung, freundlichem/kooperativen Umgang und Sichtbarkeit.» (Brand n.d.*a*: 2)

Betrachtet man die in der Literatur gemachten Vorschläge zur Lösung von Dilemmasituationen mittels Reputation genauer, dann stellt man fest, dass es eine verwirrende Vielzahl von Ansätzen gibt (Dasgupta 1988, Raub & Weesie 1990, Diekmann 1993*b*, 2004, Nowak & Sigmund 1998*a*,*b*, Suzuki & Akiyama 2005, Wedekind & Milinski 2000). Allen gemeinsam ist der Versuch, Reputation als einen Mechanismus einzuführen, welcher indirekte Reziprozität rationaler Akteure erklärt. In den in Abschnitt 3.1.2 ausgeführten Modellen war Reziprozität immer ein Resultat vorausgehender direkter Interaktion. Indirekte Reziprozität versucht diesen Mechanismus auf Situationen zu verallgemeinern, in denen die Akteure bisher noch nicht interagiert haben.

Raub & Weesie (1990) haben untersucht, wie die Effizienz in nicht kooperativen Spielen durch Reputationseffekte gesteigert werden kann. Sie zeigen am theoretischen Modell, dass Akteure einen grösseren individuellen Anreiz haben, zu kooperieren, wenn sie damit rechnen müssen, dass sonst ihre Reputation sinkt und deshalb nicht mehr mit ihnen kooperiert wird. Raub & Weesie zeigen weiter, dass dieser Effekt um so grösser ist, je stärker die Akteure untereinander Informationen über die Reputation anderer Akteure austauschen.

In einem Open Source Projekt treffen die Entwickler immer wieder auf andere Entwickler, mit denen sie bisher noch nicht direkt interagiert haben. In diesen Situationen können sie ihre Entscheidung nicht, wie beispielsweise in der Tit-For-Tat Strategie, auf direkte Erfahrung abstützen. Auch wissen sie nicht, ob sie in Zukunft wieder mit dem selben Entwickler interagieren werden. In Projekten der Grösse des Debian Projekts ist nicht sicher, dass sie in naher Zukunft noch einmal mit dem gleichen Partner interagieren werden. Dies trifft insbesondere auf Situationen wie das Beheben eines gemeldeten Fehlers zu, die im Normalfall keine wiederholte Interaktion bedingen. In solchen Situationen müssen die Entwickler auf Reputationsinformationen über das Kooperationsverhalten des Partners zurückgreifen. Sie werden dies aber nur tun, wenn sie selbst davon ausgehen, dass ihre Kooperation Dritte dazu motivieren wird, wiederum mit ihnen zu kooperieren.

Technisch wird Reputation in spieltheoretischen Modellen als «image score» modelliert. Dieser Parameter ist ein Mass dafür, wie oft ein Akteur in der Vergangenheit kooperiert hat. Nowak & Sigmund (1998*a*) haben eine Simulation von Strategien durchgeführt, welche abhängig vom image score ($-5 \leq s \leq 5$) des Partners kooperierten. Die evolutionäre Simulation hat gezeigt, dass eine Strategie, welche nur bei einem positiven oder neutralen Score kooperiert, am besten abschneidet. Wedekind & Milinski (2000) haben ein ähnliches Modell experimentell verifiziert. In ihrem Versuch haben sie bestätigt, dass das image score indirekte Reziprozität begünstigt.

Suzuki & Akiyama (2005) haben die Effekte eines image scores in grösseren Gruppen simuliert. Im Unterschied zur Simulation von Nowak & Sigmund trafen in ihrem Modell nicht jeweils zwei Individuen aufeinander, sondern die simulierten Akteure mussten sich aufgrund des mittleren image scores der Gruppenmitglieder entscheiden. Die Simulation bestand aus einem N-Personen Gefangenendilemma. Die evolutionäre Simulation hat gezeigt, dass die beste Strategie stark von der Gruppengrösse abhängig ist. In kleinen Gruppen ($N \leq 28$) waren kooperative Strategien am erfolgreichsten. In mittleren Gruppen ($28 < N \leq 45$) setzte sich neben einer kooperativen Strategie auch eine defektierende Strategie durch. Diese beiden Strategien existierten in den mittleren Gruppen nebeneinander. Keine konnte die andere zum verschwinden bringen. In grossen Gruppen ($N > 45$) setzte sich nur eine defektierende Strategie durch.

Mui et al. (2004) simulieren den Open Source Entwicklungsprozess als «Multi-Agent Modell». In jeder Runde der Simulation wird je ein Akteur als Entwickler und einer als Antragsteller (requester) ausgewählt. Der Entwickler wählt danach, ob er kooperieren und die gewünschte Entwicklung ausführen will oder nicht. Durch Kooperation steigt seine Reputation beim Antragsteller, es entstehen ihm jedoch Kosten von c. Der Antragsteller erhält einen Nutzen von b ($b > c$). Jeder Akteur verfügt über eine Strategie, mit welcher er aufgrund der Reputation des Antragstellers bestimmt, ob er kooperiert oder nicht. Zur Informationsbeschaffung über die Reputation eines Antragstellers kann ein Entwickler auch auf das Wissen der Akteure zurückgreifen, mit denen er bereits interagiert hat.

Wird die Simulation über mehrere Generationen wiederholt, an deren Ende die Akteure jeweils ihre Strategie anpassen können, so zeigt sich, dass insbesondere in kleinen Gruppen sich kooperative Strategien durchsetzen. Mit Hilfe einer Simulation ihres Modells können Mui et al. zeigen, dass Kooperation in Open Source Projekten entsteht, wenn Reputationseffekte berücksichtigt werden.

> «The emergence of voluntary action in our open source development model is then a consequence of informed discrimination.» (Mui et al. 2004: 24)

Im empirischen Teil meiner Arbeit soll es darum gehen, die im Modell gezeigten Reputationseffekte an Daten aus dem Debian Projekt zu testen. Die Reputation eines Akteurs lässt sich jedoch aus den zur Verfügung stehenden Prozessdaten nicht direkt messen. Wie jedoch oben beschrieben, entsteht eine gute Reputation durch erfolgreiche Zusammenarbeit. Meine Daten enthalten sehr genaue Informationen über die Zusammenarbeit zwischen den Entwicklern und den Arbeitseinsatz der verschiedenen Akteure für das Projekt. Daraus versuche ich die Reputation eines Akteurs zu rekonstruieren.

3.3 Hypothesen

3.3.1 Wiederholte Interaktion

Es ist zu erwarten, dass die Kooperation am grössten ist, wenn sich die beteiligten Entwickler bereits aus einer früheren Interaktion kennen. In diesem Fall können sie sich direkt an eigenen Erfahrungen orientieren und müssen nicht auf indirekte Reputationssignale, wie die geleistete Arbeit des Gegenübers, zurückgreifen. Grundsätzlich kann es sich bei einer vorausgehenden Interaktionen sowohl um eine Kooperation wie auch um eine Defektion handeln. Defektion kann aber mit dem vorliegenden Datensatz nur schwer beobachtet werden, da im Fall, dass ein Entwickler nicht kooperiert, meist auch kein Resultat aufgezeichnet wird. Somit kann beim vorliegen einer vorausgehenden Interaktion im Datensatz von einer Kooperation ausgegangen werden.

Wiederholte Interaktion ist auch ein Proxy dafür, dass Alter und Ego zusammen in einem Subprojekt aktiv sind. Unter Entwicklern, welche zusammen in kleineren Projekten aktiv sind und die sich somit gegenseitig kennen, ist eine erhöhte Kooperation wahrscheinlich.

Hypothese A Vorausgehende Kooperation zwischen zwei Entwicklern erhöht die Chance einer erfolgreichen weiteren Interaktion.

Hypothese A1 Falls sich Fehlerberichterstatter und Fehlerbeheber aus einem früheren Kontakt kennen, erhöht sich die Hazardrate der Fehlerbehebung.

3.3.2 Reputationsbildung

Ausgehend von den theoretischen Überlegungen, insbesondere der Nutzenfunktion, lassen sich Hypothesen zur Bildung von Reputation in einem Open Source Projekt bilden. Ein Entwickler, welcher neu ins System eintritt, wird anfänglich versuchen, eine gute Reputation aufzubauen. Sobald sie eine hohe Reputation aufgebaut haben, nimmt jedoch der Nutzen der durch zusätzlichen Einsatz zu gewinnenden Reputation innerhalb des Projekts ab. Ab diesem Punkt ist zu erwarten, dass sich die Akteure verstärkt darauf konzentrieren werden, aus der aufgebauten Reputation Profit zu ziehen. Insbesondere werden sie versuchen, diese auch ausserhalb des Projekts (im Arbeitsmarkt) auszunutzen.

Neben dem geringer werdenden Nutzen der zusätzlichen Reputation steigen die Opportunitätskosten. Mit der grösseren Erfahrung und der allenfalls neben dem Engagement im Open Source Projekt erlangten Hochschulbildung können Entwickler auf dem Arbeitsmarkt weit höhere Löhne erzielen. Falls die Entwicklung in der Freizeit neben der Berufsarbeit betrieben wird, ist die Chance gross, dass mit zunehmendem Alter die verfügbare Zeit aufgrund anderer Verpflichtungen, wie einer eigenen Familie, abnimmt.

Die Hypothese der steigenden Opportunitätskosten kann auch so interpretiert werden, dass Entwickler die Arbeit an einem Open Source Projekt vor allem als Investition in ihre Fähigkeiten sehen. Sobald sie nicht mehr im gleichen Ausmass neues lernen können, wird ihre Investition ins Projekt zurückgehen.

Hypothese B Entwickler investieren nach einer kurzen Anlaufphase anfänglich viel Arbeit ins Projekt und reduzieren danach ihre Investitionen.

3.3.3 Reputationsnutzen

Falls der in Hypothese B skizzierte Vorgang des Aufbaus von Reputation beim Eintritt in die Entwicklung von Open Source Software zutrifft, dann stellt sich anschliessend daran die Frage, wie die Entwickler aus der aufgebauten Reputation einen Nutzen ziehen können. Falls sich zeigen lässt, dass sie von ihrer Reputation profitieren, kann dieser Nutzen als Prämie des Reputationsaufbaus betrachtet werden.

Mit den verfügbaren Daten können nur Hypothesen zum Nutzen der Reputation innerhalb des Projekts geprüft werden. Zur Prüfung externer Reputationsprämien müssten zusätzliche Daten erhoben werden, die nicht oder nur schwer zugänglich sind. Die Hypothesen zu den Reputationsprämien fokussieren deshalb darauf, wie stark und von wem die Fehlerberichte der einzelnen Entwickler beachtet werden. Mit den vorhandenen Daten könnten zusätzlich auch Reputationseffekte auf den Mailinglisten untersucht werden. Dies würde jedoch, insbesondere aufgrund des immensen Datenbereinigungsaufwandes (siehe Abschnitt 4.2), den Rahmen dieser Arbeit sprengen.

Zudem wurde in Abschnitt 3.2.2 in Anlehnung an Granovetter (1985) die Vermutung aufgestellt, dass sich Reputation innerhalb des Projektes über ein soziales Netzwerk ausbreitet. Zentrale Projektmitglieder im Netzwerk sind besser bekannt und potentielle Interaktionspartner verfügen deshalb über zuverlässigere Reputationsinformationen. Dies erhöht die Chance einer Kooperation.

Hypothese C Die Reputation und er Status eines Entwicklers hat einen positiven Einfluss darauf, welche Beachtung seine Arbeit im Projekt findet.

- **Hypothese C1** Je höher die Reputation und der Status eines Entwicklers, desto wahrscheinlicher werden seine Fehler behoben.
- **Hypothese C2** Je höher die Reputation und der Status eines Entwicklers, desto höher ist die Hazardrate der Behebung seiner Fehler.
- **Hypothese C3** Je zentraler im sozialen Netzwerk ein Entwickler ist, desto höher ist die Hazardrate der Behebung seiner Fehler.
- **Hypothese C4** Je höher die Reputation und der Status eines Entwicklers, desto mehr Personen beachten die von ihm eingebrachten Fehlerberichte.

Hypothese D Der Status eines Entwicklers hat einen Einfluss auf den Status der Personen, welche seine Beiträge beachten.

- **Hypothese D1** Je höher der Status eines Fehlerberichterstatters, desto höher ist auch der Status desjenigen, welcher den Fehler behebt.

3.3.4 Freiwilligendilemma

Neben den Hypothesen zur Reputation lässt sich auch eine Hypothese zum Freiwilligendilemma formulieren. Wie in Abschnitt 3.1.3 ausgeführt, sinkt die Wahrscheinlichkeit der Kooperation im Freiwilligendilemma mit der Anzahl der Beteiligten.

Im Debian Projekt werden die meisten Fehler zu einem Softwarepaket gemeldet. Dieses Softwarepaket wird wiederum von einem oder mehreren Entwicklern betreut. Falls das

Freiwilligendilemma zutrifft, müssten Fehler in Paketen mit vielen verschiedenen beteiligten Entwicklern seltener behoben werden. Fehler, welche nicht ein einzelnes Softwarepaket, sondern eine Gruppe von Paketen oder das System als ganzes betreffen, können gegen verschiedene sogenannte «Pseudopakete» berichtet werden. Wer für diese Fehler zuständig ist, ist weit weniger klar definiert. Zumeist werden sie an eine Mailingliste weitergeleitet, so dass sich im Prinzip alle Abonnenten der Mailingliste zuständig fühlen sollten. Nach dem Freiwilligendilemma wäre zu erwarten, dass diese generellen Fehler weit weniger Beachtung finden.

Hypothese E Die Wahrscheinlichkeit, dass eine Arbeit ausgeführt wird, sinkt mit der Anzahl der Entwickler, welche die Arbeit ausführen könnten.

Hypothese E1 Die Hazardrate der Fehlerbehebung sinkt mit der Anzahl der Entwickler, die sich an der Entwicklung des betroffenen Softwarepakets beteiligen.

Hypothese E2 Generelle Fehler, welche nicht zu einem spezifischen Softwarepaket gemeldet werden, haben eine tiefere Hazardrate der Fehlerbehebung.

4 Daten und Methoden

Im folgenden Kapitel werde ich die Daten und die statistischen Methoden genauer vorstellen, welche im empirischen Teil meiner Arbeit verwendet werden. Zuerst werde ich darauf eingehen, welche Datenquellen benutzt wurden und wie diese Daten extrahiert wurden (4.1). Danach werde ich die Verfahren der Datenbereinigung vorstellen (4.2). Zum Abschluss des Abschnitts zu den Daten werde ich die beiden erstellten Datensätze zu den Personen (4.3.1) und zu den Fehlerberichten (4.3.2) erläutern und die wichtigsten deskriptiven Kennzahlen zu den einzelnen Variablen auflisten. Im letzten Abschnitt des Kapitels werden die angewandten statistischen Verfahren der Ereignisdatenanalyse (4.4.1) und der generalisierten linearen Regression (4.4.2) erläutert.

4.1 Datenquellen

Als Datenquellen werden die elektronischen Aufzeichnungen des Debian Projekts verwendet. Das Datenmaterial besteht ausschliesslich aus Prozessdaten, welche von den Beteiligten bei ihrer Arbeit am Debian Projekt erzeugt wurden. Beim Datenmaterial handelt es sich um eine Vollerhebung. Die Daten wurden für den gesamten jeweils zur Verfügung stehenden Zeitraum erhoben. Die elektronischen Archive des Projekts sind praktisch vollständig und weisen keine relevanten Lücken auf. Es wurde ausschliesslich öffentlich zugängliches Datenmaterial aus folgenden Quellen verwendet:

- Archive der Mailinglisten des Projekts[1]
- Fehlerdatenbank zur Meldung von Fehlern (Bugs) in Softwarepaketen[2]
- Logdateien zur geleisteten Arbeit an den einzelnen Softwarepaketen [3]
- Statistiken über die Nutzung der verschiedenen Softwarepakete[4]
- Änderungslog des «Schlüsselbundes», welcher die Verschlüsselungsschlüssel aller offiziellen Debian Entwickler enthält

Mailinglisten

Die Archive der Mailinglisten liegen als Mailboxen vor. Aus jedem Mail an eine Mailingliste wurden folgende Daten extrahiert:

- Zeitpunkt und Zeitzone

[1] http://lists.debian.org/
[2] http://bugs.debian.org/
[3] ersichtlich aus dem Archiv der Mailingliste debian-devel-changes@lists.debian.org
[4] http://popcon.debian.org

- Person, welche das Mail gesendet hat
- Umfang in Worten[5]
- Mailingliste(n) über welche das Mail versendet wurde
- Bezug zu anderen Nachrichten[6]

Fehlerdatenbank

Die Fehlerdatenbank ist komplett über Email gesteuert. Ein neuer Fehlerbericht wird mit einer Nachricht an das System eröffnet. Ein solcher Bericht bezieht sich meist auf ein Softwarepaket. Jedem Fehler ist ein Schweregrad zugeordnet. Das System kennt 7 Schweregrade von «wishlist» (Verbesserungswunsch der kein eigentlicher Fehler ist) bis zu «critical» (Fehler der das gesamte Betriebssystem unbrauchbar macht). Einem bestehenden Bericht können Zusatzinformationen hinzugefügt werden. Ein Fehler kann mit verschiedenen Bezeichnungen (sog. «Tags») versehen werden. Diese Kennzeichnen einen Fehler beispielsweise als sicherheitsrelevant oder geben an, dass das Problem zwar gelöst wurde, das entsprechende Softwarepaket jedoch noch nicht hochgeladen ist. Sobald ein Fehler behoben ist, wird dieser in der Datenbank geschlossen und archiviert. In den meisten Fällen wird ein Fehler automatisch durch das hochladen eines verbesserten Softwarepakets behoben. Die Person, welche dieses Paket hochlädt erscheint in der Fehlerdatenbank automatisch auch als diejenige Person, welche den Fehler behoben hat. Fehler können jedoch auch manuell geschlossen werden. Beispielsweise weil sie nicht durch ein verbessertes Softwarepaket behoben wurden. Während der Lebenszeit eines Fehlers können seine Eigenschaften von jedermann verändert werden. So kann der Schweregrad korrigiert, der Fehler einem anderen Paket zugewiesen, oder ein Tag hinzugefügt werden. Ein bereits als behoben markierter Fehler kann auch wieder neu geöffnet werden. Zu jedem Fehler wurden in der Auswertung folgende Daten extrahiert:

- Zeitpunkt, zu dem der Fehler gemeldet wurde
- Person, welche den Fehler meldet
- Zeitpunkt, zu welchem der Fehler behoben wurde. Falls der Fehler geschlossen und wieder eröffnet wurde, wurde der letzte Zeitpunkt, zu dem der Fehler geschlossen wurde berücksichtigt.
- Person, welche den Fehler behoben hat
- Softwarepaket, zu dem der Fehler gemeldet wurde. Falls der Fehler einem anderen Paket zugewiesen wurde. Wurde das letzte Paket berücksichtigt.
- Schweregrad des Fehlers
- Art der Fehlerbehebung: (1) Behoben durch ein verbessertes Paket; (2) manuell geschlossen; (3) Bericht wird markiert, dass es sich nicht um einen Fehler handle und deshalb nicht behoben wurde[7]; (4) das Paket wurde aus der Distribution entfernt

[5] Dabei wurden aus anderen Beiträgen zitierte Passagen so weit als möglich ignoriert.

[6] Über diese Information kann der Diskussionsverlauf rekonstruiert werden. In jeder Mail ist gespeichert, auf welche anderen Mails mit dieser geantwortet wurde. Da die Mailinglisten in der vorliegenden Arbeit jedoch nicht analysiert werden, wurde diese Information nicht weiter verwendet.

[7] Technisch gesehen werden diese Fehler von der Fehlerdatenbank nicht als geschlossen betrachtet. Sie

- Anzahl der Mails mit Zusatzinformationen im Fehlerbericht. Für jedes dieser Mails wurden zusätzlich auch die gleichen Informationen wie zu den Mails auf den Mailinglisten erhoben.

Logs der Softwarepakete

Neben der Fehlerdatenbank wurden als zweite Hauptquelle die Logs der Entwicklung der Softwarepakete analysiert. Die in der Debian Distribution enthaltene Software ist in Softwarepakete unterteilt. Der Hauptteil der Entwicklungsarbeit besteht im Unterhalt dieser Softwarepakete. Wenn ein Entwickler eine neue Version eines Pakets erstellt, führt er die notwendigen Änderungen in seiner lokalen Version des Pakets durch und testet das Paket. Jede einzelne Änderung wird im Änderungslog des jeweiligen Pakets dokumentiert. Sobald das überarbeitete Paket zufriedenstellend funktioniert, wird es vom Entwickler digital signiert und ins Debian Softwarearchiv hochgeladen. Währenddem jede Person neue Pakete erstellen oder bestehende Pakete überarbeiten kann, dürfen nur offizielle Debian Entwickler Pakete ins Archiv hochladen. Möchte eine Person, welche nicht selbst Entwickler ist, dass ihr Paket hochgeladen wird, dann muss sie einen offiziellen Entwickler finden, welcher das Paket prüft und danach stellvertretend hochlädt. Zu jedem hochgeladenen Softwarepaket wurden folgende Informationen erfasst:

- Genauer Zeitpunkt und Zeitzone
- Entwickler, welcher das Paket erstellt hat
- Entwickler, welcher das Paket signiert hat
- Weitere Personen, welche an dieser Paketversion gearbeitet haben
- Umfang des Beitrags jeder Person (Anzahl der Einträge im Änderungslog)
- Name des Softwarepakets

Die Statistiken zur Nutzung der Softwarepakete wurden zwar extrahiert und in die Datenbank abgespeichert, jedoch nicht für die Auswertung verwendet, da sie nur für einen ungenügenden Zeitraum verfügbar waren. Auch konnten in ersten Analysen kein relevanter Zusammenhang zur Fehlerbehebung festgestellt werden. Das Änderungslog des «Schlüsselbundes» wurde nur verwendet, um die offiziellen Debian Entwickler zu identifizieren.

Die Daten wurden für den Zeitraum vom April 1995[8] bis zum November 2007 erhoben. Die Rohdaten wurden aus dem Archiv des Debian Projekts heruntergeladen. Danach wurden die einzelnen Beiträge mit Hilfe mehrerer eigens dazu entwickelter Skripte ausgewertet und in einer Datenbank abgespeichert. Dabei wurden Rohdaten im Umfang von 38

sind lediglich mit dem «Tag» «wontfix» markiert. Für die Auswertung wurden diese Fehler als geschlossen betrachtet. Als Zeitpunkt der Fehlerbehebung gilt der Zeitpunkt zu dem das «wontfix» «Tag» hinzugefügt wurde.

[8] Der genaue Zeitraum variiert je nach Datenquelle. Die Daten zur Arbeit an den einzelnen Softwarepaketen sind erst ab 1996 verfügbar und die Statistiken zur Nutzung der Software erst ab 2004. Auch wurden zahlreiche Mailinglisten erst später eingerichtet.

Gigabyte[9] bearbeitet. Zur Bewältigung dieser riesigen Datenmengen wurde die Extraktion der Informationen auf dem Linux Cluster der Universität Bern (UBELIX) durchgeführt.

Um die Auswertung zu vereinfachen, wurden die so erfassten Daten pro Person und Monat zusammengefasst und diese aggregierten Daten wiederum in der Datenbank abgespeichert. Diese Aggregation wurde wiederum mit einem speziell für diesen Zweck entwickelten Skript ausgeführt. Die Berechnung wurde parallel auf 40 Knoten des Linux Clusters in ca. 12 Stunden ausgeführt. Dies ergab den Datensatz zur Entwickleraktivität. Mit einem analogen Verfahren wurde auch der Datensatz zu den Fehlerberichten erstellt.

Mit diesem Datenmaterial ist es möglich, den grössten Teil der Arbeit am Debian Projekt zu erfassen. Es gibt jedoch einige Bereiche, welche nicht erfasst werden. Dazu gehören der private Mailaustausch zwischen Entwicklern, welcher weder über eine Mailingliste noch über die Fehlerdatenbank geht. Weiter die Arbeiten, welche einige Entwickler an der Projekt Infrastruktur verrichten, wie den Betrieb der verschiedenen Servern, die für die Arbeit der Entwickler wichtig sind. Ausserdem finden Diskussionen auch ausserhalb der Mailinglisten im IRC (Chatsystem) statt. Ich gehe jedoch davon aus, dass alle diese Bereiche im Vergleich zu den von den Daten erfassten Bereichen relativ geringfügig sind. Insbesondere im Fall der Diskussionen im IRC ist es wahrscheinlich, dass sich Spuren dieser Arbeit auch in den Daten finden werden. Beispielsweise durch eine Fortsetzung der Diskussion auf einer Mailingliste oder durch die Behebung eines im Chat besprochenen Fehlers.

4.2 Datenbereinigung

Die nach der automatischen Aufbereitung in der Datenbank abgespeicherten Beiträge mussten danach noch bereinigt werden. Diese Datenbereinigung konnte nur zu einem kleinen Teil automatisiert werden. Es wurden folgende Bereinigungen vorgenommen:

- Eliminierungen von ungültigen Beträgen: Spam, Beiträge, welche nur generischen Mailadressen zugeordnet werden konnten (beispielsweise der generischen Kontaktadresse security@debian.org)
- Eliminierung von offensichtlichen Fehlzuordnungen
- Verknüpfung von Beiträgen, welche von der gleichen Person unter Verwendung unterschiedlicher Mailadressen geleistet wurden

Insbesondere der letzte Schritt war sehr Zeitaufwändig und konnte nur zu einem kleinen Grad automatisiert werden. Deshalb wurde nicht der ganze Datensatz vollständig bereinigt, sondern nur diejenigen Mailadressen, welche mindestens einen Fehlerbericht eingesandt haben oder sich an der Arbeit an einem Softwarepaket beteiligten. In der Folge wurden auch alle Auswertungen auf diesen Teildatensatz beschränkt.

[9] 23 GB Fehlerberichte, 13 GB Mailinglistendaten, 704 MB Nutzungsstatistiken und 1 GB Logdateien von Softwarepaketen.

Tabelle 4.1: Übersicht über die Anzahl Personen im Datensatz

	N	**(aktiv)**			
Gesamter Datensatz	236620		100%		
Bereinigter Datensatz	32538	(7206)	14%	100%	
Contributors	30027	(5647)	13%	92%	
Entwickler	2512	(1559)	1%	8%	100%
Einfache Entwickler	1181	(745)	<1%	4%	46%
Offizielle Entwickler	827	(390)	<1%	3%	35%
Kernentwickler	503	(424)	<1%	2%	20%

Anmerkung: In Klammern jeweils die Zahl der Personen, welche zum Ende der Untersuchung noch aktiv waren. Die Prozentzahlen beziehen sich auf den Anteil der jeweiligen Kategorie am gesamten Datensatz, am bereinigten Datensatz bzw. an den Entwicklern.

Zur Verknüpfung der Mailadressen wurden verschiedenste Kriterien angewandt. Dabei wurde so weit als möglich versucht, automatisiert nach möglichen Verknüpfungen zu suchen. Dazu wurden die Namen und Mailadressen zuerst normalisiert und in einzelne Teile (Vorname, Nachname, Benutzername, Domain, ...) aufgeschlüsselt. Neben Ähnlichkeiten in Namen und Email wurden als weitere Indikatoren der Zeitpunkt der Beiträge, die Server über welche die Mails versendet wurden (Message-ID) und die Mailinglisten und Pakete, zu denen die Beiträge zugeordnet waren, verwendet. Die so gefunden Verknüpfungsvorschläge wurden dann noch manuell verifiziert. Im Zweifelsfall wurden die Beiträge nicht verknüpft.

4.3 Datensätze

Zur weiteren Analyse wurden aus der Datenbank zwei Datensätze gewonnen. Erstens ein Datensatz zur Aktivität der Personen im Projekt und zweitens ein Datensatz über alle Fehlerberichte.

4.3.1 Datensatz Entwickleraktivität

In diesem Datensatz wurden die einer Person zugeordneten Beiträge (Mails, Fehlerberichte, Paketarbeit, ...) jeweils für jeden Monat summiert. Jede Person wurde nach dem Ausmass ihrer Projektbeteiligung einer der in Abschnitt 2.2.2 definierten Statuskategorien zugeordnet.

Tabelle 4.1 gibt eine Übersicht über die Anzahl der Personen in den verschiedenen Kategorien. Die grosse Differenz zwischen dem gesamten Datensatz und dem bereinigten Datensatz ergibt sich daraus, dass nur die Personen bereinigt wurden, welche sich an einem Fehlerbericht oder der Paketentwicklung beteiligten. Der weitaus grösste Teil der Personen ist jedoch nur in den Mailinglisten aktiv. Diese Personen werden nicht weiter betrachtet. Aus den erhobenen Daten wurden 12 personenbezogene Variablen gewonnen.

Tabelle 4.2: Übersicht über die personenbezogenen Variablen

Name	Beschreibung
bugs sub	Anzahl der Fehlerberichte
bug act m	Anzahl der Mails mit Zusatzinformationen an einen bestehenden Fehlerbericht
bug act w	Anzahl der Wörter in einem Email an einen Fehlerbericht
bug fix	Anzahl der behobenen Fehler
patch	Anzahl der Patches zu Fehlerberichten
mails	Anzahl der Mails auf einer Mailingliste
words	Anzahl der Wörter in den Mails an eine Mailingliste
core m	Anzahl der Mails an eine zentrale, von vielen Entwicklern gelesene Mailingliste
core w	Anzahl der Wörter in den Mails an zentrale Mailinglisten
uploads	Anzahl der erstellten Paketversionen
upl sig	Anzahl der signierten Paketversionen
devel	Anzahl der Änderungen an Paketen

Anmerkung: Ein Patch ist ein Stück Programmcode, welcher einen vorhandenen Fehler behebt. Ein Patch kann von jeder beliebigen Person zu einem Fehlerbericht hinzugefügt werden. Ein Patch kann als ein Lösungsvorschlag zu einem bestimmten Fehler betrachtet werden.

Für jede Variable wurde jeweils das Total über den gesamten Untersuchungszeitraum und die Zwischensumme für jeden Monat berechnet. Die Daten entsprechen somit einem Paneldesign mit 152 Wellen im Abstand von jeweils einem Monat. Neben dem absoluten Zeitpunkt eines Beitrags wurde jeweils auch die relative Zeit seit dem Eintritt ins Projekt einer Person berechnet. Mit Hilfe dieser relativen Zeit kann die Entwicklertätigkeit im Zeitverlauf analysiert werden.

Aus dem Hauptdatensatz zur Entwickleraktivität wurden einige weitere Datensätze gewonnen. So wurden ein Datensatz zu den ununterbrochenen Perioden der Aktivität für jeden Entwickler erstellt. Dieser enthält die Dauer und Anzahl der Aktivitätsperioden und die Dauer und Anzahl der dazwischen liegenden Perioden, in denen für diese Person keine Aktivität registriert wurde. Weiter wurde ein nicht personenbezogener Datensatz zum Zeitpunkt der jeweiligen Aktivitäten generiert. Dieser enthält die genaue Uhrzeit und die Zeitzone jedes Beitrags.

Tabelle 4.3: Rangkorrelation (Spearman's ρ) der Variablen im Datensatz Entwickleraktivität

	bug sub	**bug act m**	**bug act w**	**bug fix**	**patch**	**mails**	**words**	**core m**	**core w**	**uploads**	**upl sig**
bug sub	1										
bug act m	**0.89**	1									
bug act w	**0.78**	**0.84**	1								
bug fix	0.42	0.49	0.44	1							
patch	0.32	0.33	0.31	0.52	1						
mails	0.49	0.51	0.44	0.40	0.29	1					
words	0.49	0.51	0.44	0.40	0.29	**1.00**	1				
core m	0.42	0.42	0.38	0.52	0.42	0.55	0.55	1			
core w	0.42	0.43	0.38	0.51	0.42	0.55	0.55	**1.00**	1		
uploads	0.33	0.39	0.35	**0.74**	0.52	0.39	0.39	0.56	0.55	1	
upl sig	0.29	0.32	0.29	0.57	0.48	0.33	0.33	0.54	0.54	**0.75**	1
devel	0.32	0.39	0.35	**0.73**	0.52	0.38	0.38	0.54	0.54	**0.97**	**0.72**

Anmerkung: Da viele der Variablen sehr weit von einer Normalverteilung entfernt sind, sind die Voraussetzungen für die übliche Pearson-Korrelation nicht erfüllt. Deshalb wurde eine Rangkorrelation verwendet. Die Ergebnisse unterscheiden sich jedoch höchsten im hinteren Hunderstelbereich.

Tabelle 4.2 gibt einen Überblick über die Variablen. Teilweise wurden zur gleichen Aktivität mehrere Kennzahlen erhoben. So wurde zu den Mails jeweils die Anzahl der Nachrichten wie auch die Anzahl der Wörter erfasst. Auch wurden zur Entwicklungsarbeit die Anzahl der einzelnen Änderungen, die Anzahl der aktualisierten Pakete und die Anzahl der signierten Pakete erfasst. Tabelle 4.3 zeigt jedoch, dass die Korrelation zwischen diesen Werten hoch ist (fett gedruckte Werte). Auch die Anzahl an Fehlerberichte gesendete Mails mit Zusatzinformationen korreliert stark mit der Anzahl der gemeldeten Fehler. Aus dieser hohen Korrelation lässt sich schliessen, dass diese Variablen jeweils das gleiche oder ein sehr ähnliches Grundkonstrukt messen. Deshalb werden in den weiteren Auswertungen für die Fehlerberichtsaktivität, die Aktivität auf den Mailinglisten und die Aktivität in der Entwicklung nur die in Tabelle 4.2 fett gedruckten Variablen berücksichtigt. Bei mehreren zur Auswahl stehenden Variablen wurde jeweils diejenige gewählt, welche den zu messenden Beitrag am genausten erfasst.

Tabelle 4.4 gibt eine Übersicht über die Werte und die Verteilung der personenbezogenen Variablen. Alle Variablen zeichnen sich durch eine sehr breite Streuung aus. Auch wenn nur die Werte für die einzelnen Subgruppen betrachtet werden, ist die Standardabweichung eher noch grösser. Die grosse Streuung rührt insbesondere daher, dass alle Variablen extrem rechtsschief sind. Nur sehr wenige Personen leisten einen grossen Teil der Arbeit. Als Mass für diese ungleiche Verteilung ist jeweils der Gini-Koeffizient angegeben.

4.3.2 Datensatz Fehlerberichte

Zur Verifizierung der Reputationshypothesen und der Hypothesen zum Freiwilligendilemma wurde ein Datensatz mit allen Fehlerberichten der Fehlerdatenbank erstellt.

Fehlerberichterstatter und Fehlerbeheber

Zu den Fehlerberichterstattern und den Fehlerbehebern wurden der Status nach der Kategorisierung in Abschnitt 2.2.2, die bisherige Projektaktivität, das Ausmass der Zusammenarbeit mit anderen Entwicklern und die Zeitdauer seit dem ersten Beitrag erfasst. Alle Variablen beziehen sich auf den Zeitpunkt zu dem der Fehlerbericht eingesandt wurde. Die Variablen zum Fehlerbeheber sind nur für Fehler verfügbar, welche bereits behoben sind. Für alle zensierten Fälle sind sie nicht verfügbar, da der Fehlerbeheber noch nicht bekannt ist. Dies führt dazu, dass aus allen Modellen, in die Eigenschaften des Fehlerbehebers aufgenommen werden, die zensierten Fälle als missing values verschwinden.

Status Der Status entspricht der Definition in Abschnitt 2.2.2. Problematisch ist dabei die Zuordnung zum Status der Kernentwickler. Die Kernentwickler sind als die 20% aktivsten Entwickler definiert. Die absolute Höhe dieser Schwelle ändert sich aber über die Zeit. Diese Veränderung wurde jedoch nicht berücksichtigt. Als Kernentwickler wurden diejenigen Entwickler codiert, deren Aktivität bis zum Zeitpunkt des Fehlerberichts über dem 0.8 Quantil der Aktivität zum Untersuchungsende lag. Der Status ist wohl die von anderen Projektmitgliedern am direktesten wahrnehmbare Reputationsvariable. Offizielle Entwickler sind beispielsweise an ihrer Mailadresse erkennbar. Ob jemand sich an der Entwicklungsarbeit beteiligt, kann

Tabelle 4.4: Deskriptive Statistik zum Datensatz Entwickleraktivität nach Entwicklerstatus

	bugs sub	bugs fix	patch	words	core w	devl
Alle Personen (N=32'539)						
Werte > 0	32'253	3'850	1'313	12'908	3'935	2'512
Median	2	5	3	1'117	611	50
Mittelwert	13	90	9	11'740	7'371	243
Std. Abweichung	83	305	23	83'268	60'739	627
Minimalwert	1	1	1	1	1	1
Maximalwert	5'182	7'168	302	5'035'335	3'342'730	16'735
Gini Koeffizient	0.84	0.85	0.70	0.87	0.88	0.77
Contributor (N=30'027)						
Werte > 0	29'934	1'613	363	10'624	2'176	-
Median	1	1	1	793	296	-
Mittelwert	6	4	3	5'364	1'344	-
Std. Abweichung	38	15	6	19'761	4'323	-
Minimalwert	1	1	1	1	1	-
Maximalwert	3'877	3'870	82	861'636	94'103	-
Gini Koeffizient	0.74	0.64	0.53	0.81	0.77	-
einfache Entwickler (N=1'181)						
Werte > 0	1'022	975	286	976	511	1'181
Median	12	7	3	2'494	412	12
Mittelwert	43	19	7	15'647	2'699	30
Std. Abweichung	112	42	19	63'441	14'870	44
Minimalwert	1	1	1	8	6	1
Maximalwert	1'276	720	291	888'601	294'186	272
Gini Koeffizient	0.74	0.67	0.65	0.83	0.84	0.65
offizielle Entwickler (N=828)						
Werte > 0	793	759	275	805	753	828
Median	24	43	3	7'585	1'866	83
Mittelwert	56	62	6	24'171	7'464	103
Std. Abweichung	93	65	8	52'299	21'732	80
Minimalwert	1	1	1	16	5	1
Maximalwert	1'044	597	56	621'296	347'460	287
Gini Koeffizient	0.64	0.67	0.57	0.72	0.76	0.44
Kernentwickler (N=503)						
Werte > 0	503	503	389	503	495	503
Median	143	336	9	36'332	9'418	653
Mittelwert	287	542	20	118'948	38'549	976
Std. Abweichung	480	681	36	380'991	164'906	1'128
Minimalwert	5	1	1	32	17	288
Maximalwert	5'182	7'168	302	5'035'335	3'342'730	16'735
Gini Koeffizient	0.61	0.49	0.66	0.74	0.77	0.43

Anmerkung: Es werden nur diejenigen Variablen aufgelistet, welche im weiteren Verlauf berücksichtigt werden. Die nicht aufgeführten Variablen wurden wegen der hohen Korrelation mit anderen Variablen ausgeschlossen. Der Wert der Variable ist jeweils die Summe der Beiträge einer Person über den gesamten Untersuchungszeitraum.

durch sehr einfache Recherchen herausgefunden werden und ein grosser Teil der Kernentwickler dürfte den anderen Entwicklern namentlich bekannt sein. In die Regressionsrechnungen wird der Status mit drei Dummy-Variablen für einfache Entwickler, offizielle Entwickler und Kernentwickler integriert. Für die Contributoren wird keine Dummy-Variable gebildet. Sie bilden den Referenzpunkt der Schätzungen der anderen Dummy-Variablen.

Aktivität Die Aktivität setzt sich aus fünf Einzelvariablen zusammen. Sie enthält (1) die eingesandten Fehler (2) die behobenen Fehler (3) die eingesandten Patches (4) die Anzahl der Worte auf den Mailinglisten und (5) die Paketarbeit. Alle diese einzelnen Variablen wurden logarithmiert, auf das Intervall 0 bis 1 normiert und danach addiert. Daraus ergibt sich ein Aktivitätsindex mit Werten zwischen 0 und 5, welcher zumindest im mittleren Wertebereich annähernd normalverteilt ist. Der Aktivitätsindex misst theoretisch die ins Projekt eingebrachte Arbeitsleistung genauer als der Status einer Person. Allerdings kann die Aktivität eines Projektmitglieds wesentlich weniger gut von den anderen Projektmitgliedern im vollen Umfang wahrgenommen werden, da das Debian Projekt zu gross ist, dass alle Mitglieder einen Überblick über die Arbeit aller anderen Mitglieder haben können. Die Aktivität kann als Proxy für das in Abschnitt 3.2.2 eingeführte «image score» betrachtet werden.

Zusammenarbeit Der Umfang der Zusammenarbeit mit anderen Projektmitgliedern wird sowohl für die Fehlerberichte, wie auch für die Paketarbeit erfasst. Die Variable misst die Anzahl der weiteren Personen, welche an Fehlerberichten bzw. Paketen beteiligt sind, an welchen die Person gearbeitet hat. Im Gegensatz zum Aktivitätsindex, welcher die Aktivität erfasst, ist die Zusammenarbeit ein Proxy für die Anzahl der Personen, mit denen diese Person bereits in Kontakt trat. Die Zusammenarbeit kann auch als Netzwerkmass aufgefasst werden. Aus der Zusammenarbeit zwischen den einzelnen Projektmitgliedern kann ein Kooperationsnetzwerk gebildet werden. Zusammenarbeit bei der Behebung eines Fehlers oder an einem Softwarepaket sind dabei die Kanten des Netzwerks zwischen den Projektmitgliedern. Die Kooperationsvariable ist in dieser Betrachtung der Degree einer Person in diesem Netzwerk und damit ein Mass für die Zentralität im Netzwerk. Die logarithmierten Zusammenarbeitsvariablen korrelieren sehr stark ($\rho > 0.9$) mit dem Aktivitätsindex. Im Unterschied zum Aktivitätsindex messen die Kooperationsvariablen aber nicht den Umfang der Arbeitsleistung, sondern die Anzahl Projektmitglieder, zu welchen eine Person Kontakt hatte.

Dauer der Projektbeteiligung Zeitdauer zwischen dem ersten Beitrag bis zum Zeitpunkt zu dem der Fehler gemeldet wird in Jahren. Diese Variable dient als Proxy für die Erfahrung einer Person. Ich nehme an, dass Personen mit zunehmender Dauer ihrer Projektbeteiligung eine grössere Erfahrung über die Prozesse des Projektes und den Umgang mit Fehlern haben. Auch lernen sie mit der Zeit, einfacher zu bearbeitende Fehlerberichte zu verfassen. Allerdings korreliert diese Variable sowohl mit der Aktivität ($\rho = 0.63$), als auch mit der Kooperation ($\rho = 0.48$ bzw $\rho = 0.42$).

Medianzeit Fehlerbehebung Median der bisher vom Fehlerbeheber behobenen Fehler in Monaten. Diese Variable ist jeweils für den ersten Fehler, welchen eine Person behebt, nicht definiert. Diese Variable wird als Kontrollvariable verwendet. Sie soll die individuellen Unterschiede, wie rasch verschiedene Entwickler auf Fehler reagieren, ausgleichen. Insbesondere für weniger aktive Entwickler basiert diese Variable jedoch auf relativ wenigen Fehlern. Im Median werden ca. 200 Fehler berücksichtigt. Deshalb ist die Variable für die ersten von einem Entwickler behobenen Fehler auch sehr instabil.

Vergangene Interaktion Dichotome Variable, welche anzeigt, ob der Fehlerberichterstatter und der Fehlerbeheber früher bereits direkten Kontakt zueinander hatten. Als vergangene Interaktionen werden Fehlerberichte, an denen beide beteiligt waren, Pakete, an denen beide gearbeitet haben und «sponsoren» (signieren und hochladen) eines durch die andere Person erstellten Paketes gezählt. Diese Variable ist nur für diejenigen Fehlerberichte verfügbar, welche bereits behoben wurden. Andernfalls ist der Fehlerbeheber noch nicht bekannt und es kann deshalb auch keine vergangene Interaktion ermittelt werden. Deshalb wird diese Variable nur in denjenigen Modelle verwendet, welche auch die Variablen zum Fehlerbeheber enthalten. Das Ausmass der vergangenen Interaktion ist erstaunlich hoch. So haben 42% (46% im reduzierten Datensatz) der Fehlerberichterstatter und Fehlerbehebenden bereits früher einmal miteinander interagiert. Allerdings ist die hier verwendete Definition der vergangene Interaktion auch sehr breit. Es werden auch sehr lockere Kontakte als vergangene Interaktion gewertet.

Fehlerbericht

Zeit bis zur Behebung («Überlebenszeit») Zeitdauer in Tagen bis ein Fehler behoben wurde. Diese Variable ist für alle Fehler rechtszensiert, welche bis zum Ende der Erhebungsperiode nicht behoben wurden. Zur Prüfung der Hypothesen, welche sich auf die Zeitdauer bis zur Fehlerbehebung beziehen ist dies die abhängige Variable. Im Modell zur Überprüfung der Hypothese zur Anzahl der Reaktionen auf einen Fehlerbericht wird sie aber auch als Kontrollvariable verwendet, da mit zunehmender Zeit auch die Wahrscheinlichkeit zunimmt, dass sich mehr Personen an einem Fehler beteiligen.

Schweregrad Jedem Fehler wird einer von sieben Schweregraden zugeordnet: wishlist, minor, normal, important, serious, grave, critical. Fehler mit dem Grad wishlist sind keine Fehler, im engeren Sinn, sondern Wünsche für zusätzliche Funktionen. Standardmässig hat ein Fehler den Grad normal. Fehler der Grade serious, grave und critical gelten als «release critical» und müssen vor einem Release behoben werden. Da der Schweregrad eines Fehler nach dem einsenden noch verändert werden kann, wurde jeweils der Schweregrad zum Zeitpunkt der Fehlerbehebung bzw. der Zensierung aufgezeichnet. Da der Schweregrad nicht a priori Intervallskalenniveau aufweist, wird er als ordinal skaliert betrachtet und wie die Kategorie mit Dummy-Variablen in die Regressionsrechnungen eingefügt.

Anzahl Paketbetreuer Anzahl der Entwickler, die in den 365 Tagen vor dem Fehlerbericht an dem betroffenen Paket gearbeitet haben. Diese Variable ist nur für ca. 2/3 der Pakete verfügbar, da die Arbeit der Entwickler auf Basis der Quellcodepakete erfasst wurde und die meisten Fehler jedoch gegen die daraus generierten Binärpakete gemeldet werden. Deshalb konnte diese Variable nur für Fehler erhoben werden, bei denen der Name des Quellcodepakets und des Binärpakets übereinstimmen. Mit dieser Variable soll die Annahme des Freiwilligendilemmas überprüft werden, dass die Chance zur Erbringung einer Leistung mit der Zahl der verfügbaren Freiwilligen sinkt.

Anzahl der Beteiligten Anzahl der an einem Fehlerbericht beteiligten Personen. Zur Überprüfung der Hypothese, wonach sich mit zunehmender Reputation des Fehlerberichterstatters mehr Personen an einem Fehler beteiligen, wird diese Variable als abhängige Variable verwendet.

Pseudopaket Neben den Softwarepaketen können Fehler auch zu mehreren sogenannten Pseudopaketen gemeldet werden. Diese sind für Fehler gedacht, welche nicht einem Paket zugeordnet werden können. So gibt es beispielsweise Pseudopakete für generelle Fehler, welche die gesamte Distribution betreffen, für Fehler im Installationsprozess oder bei Upgrades und für Fehler an der Webseite des Debian Projekts. Fehlerberichte zu Pseudopaketen werden nicht an eine Person oder einen kleinen Personenkreis, sondern an eine «zuständige» Mailingliste weitergeleitet. Alle Fehler wurden bezüglich der Pseudopakete in drei Gruppen aufgeteilt: Fehler in normalen Pakten, Fehler in Pseudopaketen, für die eine mehr oder weniger definiertes Team zuständig ist und in Fehler für die alle Entwickler gemeinsam zuständig sind. Zudem wurden Fehler zu den Pseudopaketen des technischen Komitees (tech-ctte) und der «Work-needing and prospecive packages (wnpp)» komplett aus dem Datensatz ausgeschlossen. Für diese Fehler gelten spezifische Regeln und es handelt sich dabei nicht zwingend um Fehler im eigentlichen Sinn. Diese Fehlerberichte dienen mehr dazu, einen Prozess oder den Zustand eines Paketes zu dokumentieren.

Anzahl Worte im Fehlerbericht Der Umfang des Fehlerberichts wird als Kontrollvariable verwendet. Mit ihm soll die Präzision der Fehlerbeschreibung und der Umfang der zum Fehler verfügbaren Information erfasst werden. Ich gehe davon aus, dass präziser analysierte Fehler rascher behoben werden.

Art der Fehlerbehebung Ein Fehler kann auf vier verschiedene Arten behoben werden. Mit Abstand am häufigsten wird ein Fehler durch ein verbessertes Softwarepaket behoben. Daneben kann ein Fehler aber auch per Email mit einer Erklärung behoben werden. Bei einem grossen Teil der so geschlossenen Fehler handelt es sich um Probleme, die vom zuständigen Entwickler nicht als Fehler im Paket angesehen werden. Zudem kann ein Fehler als «wontfix» markiert werden. Dies bedeutet, dass der Paketbetreuer einen Fehler zwar anerkennt, er diesen aber nicht beheben möchte. Beispielsweise weil die Behebung zu grosse Änderungen am Paket erfordern würde oder weil durch die Behebung andere als wichtiger erachtete Funktionen verloren

Tabelle 4.5: Bereinigungsschritte und Subsample im Datensatz Fehlerberichte

	N	**zensiert**
unbereinigter Datensatz	419'472	75'920
negative Überlebenszeit	331	3
Berichterstatter = Fehlerbeheber	25'515	0
vor 1. Januar 1999	83'143	10'095
technisches Komitee und «WNPP»	17'810	2'184
bereinigter Datensatz	305'342	63'681
nicht durch neues Paket behoben	88'689	0
mehr als zwei Beteiligte	107'731	17'816
«komplexe» Fehler	57'553	11'207
reduzierter Datensatz	131'382	41'465

gehen würden. Als letzte Variante kann ein Fehler durch die Entfernung eines Pakets aus dem Debian Archiv behoben werden.

Auch wenn einige der Variablen in Tagen, Monaten oder Jahren gemessen wurden, so wurden trotzdem alle Zeitspannen mit einer Auflösung von einer Sekunde[10] gemessen. Die Zeit wurde jeweils ohne Verlust an Genauigkeit in eine Einheit transformiert, welche die Interpretation der Ergebnisse vereinfacht, da die Effekte bei einer Messung in Sekunden für jede einzelne Sekunde verschwindend klein wären.

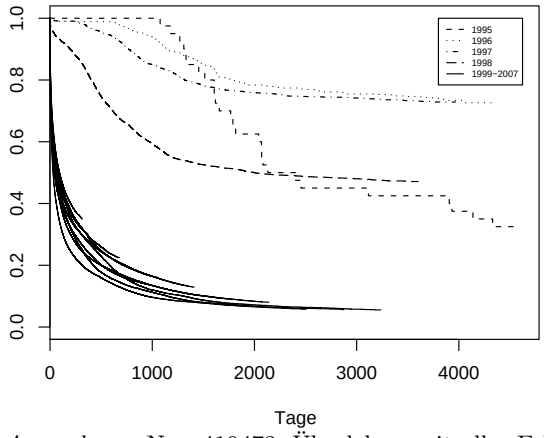
Aufgrund ihrer sehr schiefen Verteilung wurden die Variablen zur Zusammenarbeit und die Worte im Fehlerbericht jeweils logarithmiert in die Modelle eingefügt. Damit wird verhindert, dass sehr grosse Werte überbewertet werden. Die Logarithmierung modelliert einen abnehmenden Grenznutzen der jeweiligen Variablen. Falls eine Person bereits mit sehr vielen anderen interagiert hat, dann ist der Nutzen einer zusätzlichen Interaktion kleiner, als wenn es sich dabei um die erste Interaktion handelt.

Aus dem Datensatz wurden vor der Analyse alle Fehlerberichte mit einer negativen Überlebenszeit, die Fehlerberichte, bei denen die gleiche Person sowohl den Fehler gemeldet als auch behoben hat und alle Fehlerberichte, welche vor dem 1. Januar 1999 eingesandt wurden, entfernt. Zudem wurde ein Subsample erstellt, welches nur die «einfachen» Fehlerberichte enthält. Tabelle 4.5 gibt einen Überblick über die einzelnen Bereinigunsschritte und das Subsample.

Eine sehr kleine Zahl der Fehlerberichte hat eine negative Überlebenszeit. Diese kommt wahrscheinlich daher, dass die Uhr des Computers, auf dem der Bericht versandt wurde, eine geringfügige Abweichung hatte. Grössere Abweichungen wurden durch den Vergleich

[10] Es ist klar, dass eine solche präzise Auflösung für die in dieser Arbeit untersuchten Phänomene völlig unnötig ist. Da die Zeit jedoch bereits sekundengenau im Ausgangsmaterial vorhanden war, gibt es keinen Grund, durch willkürliches Runden auf diese Genauigkeit zu verzichten. Auch sind die Zeiten mit gewissen Ungenauigkeiten behaftet, da die Verzögerung bei der Verarbeitung der Fehlerberichte auf den Servern nicht berücksichtigt wurde. Die effektive Genauigkeit liegt wohl im Bereich von Stunden.

Abbildung 4.1: Überlebenskurven nach Einsendezeitpunkt



Anmerkung: N = 419472; Überlebenszeit aller Fehlerberichte unterteilt nach dem Jahr des Einsendezeitpunktes.

des Sendezeitpunktes mit dem Zeitpunkt, zu dem der Fehler auf dem Server des Debian Projekts eintraf, erkannt. Sehr kleine Abweichungen kombiniert mit einem Fehler, der sehr schnell behoben wurde, konnten so aber nicht erkannt werden.

Es kommt immer wieder vor, dass ein Fehler von der gleichen Person eingesandt wird, die diesen auch behebt. Dafür sind mehrere Gründe denkbar. Am wahrscheinlichsten ist, dass der Einsender des Fehlers selbst bemerkt hat, dass es sich bei seinem Problem gar nicht um einen Softwarefehler handelt. Für die Analysen meiner Hypothesen sind diese Berichte uninteressant und werden deshalb ausgeschlossen.

Eine genauere Analyse der Überlebensfunktion nach Einsendezeitpunkt des Fehlers hat gezeigt, dass sich Fehler, die vor 1999 eingesandt wurden, wesentlich von den später eingesandten Fehlern unterscheiden. Sie hatten eine wesentlich tiefere Hazardrate. Die wahrscheinlichste Ursache für diesen Umstand ist, dass nicht mehr alle Fehler aus dem Zeitraum vor 1999 in der Fehlerdatenbank vorhanden sind. Darauf deutet auch hin, dass die Identifikationsnummern der Fehler im unteren Bereich nicht fortlaufend sind. Sehr wahrscheinlich wurden bei einer Migration des Fehlerdatenbanksystems nur die zu diesem Zeitpunkt noch nicht behobenen Fehler migriert. Grafik 4.1 zeigt die Überlebenskurven aufgeteilt nach Einsendezeitpunkt des Fehlers. Zudem wurden, wie bereits erwähnt, alle Fehler zu den Pseudopaketen des technischen Komitees und der «work-needing and prospective packages» aus dem Datensatz entfernt.

Zusätzlich zum bereinigten Datensatz wurde ein Subsample mit «einfachen» Fehlerberichten erstellt. Dabei wurde versucht, alle Fehler, welche das Ergebnis zu stark verfälschen könnten aus dem Datensatz zu entfernen. Ich werde diesen Datensatz im Folgenden als «reduzierten Datensatz» bezeichnen. Dazu wurden folgende Kriterien angewandt:

1. Es wurden nur Fehler berücksichtigt, welche durch ein verbessertes Paket behoben wurden. Wie bereits weiter oben erläutert, handelt es sich bei den anderen

Fehlerberichten meist nicht um Fehler im engeren Sinn.

2. Es wurden nur Fehler berücksichtigt, an deren Lösung inklusive dem Fehlerberichterstatter nicht mehr als zwei Personen beteiligt waren. So kann sichergestellt werden, dass die Interaktion zwischen dem Fehlerberichterstatter und dem -beheber nicht durch Einflüsse weiterer Personen verfälscht wird. Fehlerberichte, welche noch von keiner weiteren Person ausser dem Fehlerberichterstatter beachtet wurden, befinden sich auch im Datensatz.

3. Es wurden alle «komplexen» Fehler ausgeschlossen. Die Fehlerdatenbank erlaubt eine Reihe von Manipulationen an Fehlern, welche eine genaue Bestimmung der Zeit bis zur Behebung des Fehler erschweren. So kann ein Fehler geschlossen und später wieder neu eröffnet werden, weil er doch noch nicht (vollständig) behoben war. Auch können Fehler dupliziert und so mehreren Paketen zugeordnet werden. Alle Fehler, an denen eine dieser Manipulationen vorgenommen wurde, wurden bereits bei der Auswertung der Rohdaten als «komplex» markiert.

Deskriptive Statistiken zu allen Variablen dieses Datensatzes befinden sich im Anhang A.1. Dort sind auch die einzelnen Variablen aufgeführt, welche zum Aktivitätsindex zusammengefasst wurden.

4.4 Statistische Modelle

Im folgenden werde ich die im empirischen Teil der Arbeit verwendeten statistischen Modelle kurz vorstellen. Ich werde dabei das Schwergewicht auf die zentralen Annahmen der jeweiligen Regressionsmodelle legen und ausführen, wie die geschätzten Parameter zu interpretieren sind. Ich werde jedoch weder eine detaillierte Einführung in die statistischen Grundlagen geben, noch auf die Verfahren eingehen, mit welchen die Modellparameter geschätzt werden. Dazu sei auf die einschlägige Literatur verwiesen.

Alle statistischen Analysen in der vorliegenden Arbeit wurden mit dem Open Source Statistikpaket R (R Development Core Team 2008) durchgeführt. R bietet neben einem Kern, der alle üblichen statistischen Verfahren enthält, eine Fülle von Zusatzpaketen für alle verschiedenste Anwendungsbereiche. R hat einen Funktionsumfang, der mit bekannten kommerziellen Statistikpaketen wie Stata und SPSS vergleichbar ist. Für die Poisson- und Logit-Regressionen wurde das Zusatzpaket «Design» (Harrell 2008) verwendet. Zur Ereignisdatenanalyse das Zusatzpaket «Survival» (Therneau & Lumley 2008).

4.4.1 Ereignisdatenanalyse

Die Ereignisdatenanalyse[11] wird verwendet, um Prozesse in ihrem Zeitverlauf zu analysieren. Die Fehlerbehebung ist ein Prozess, bei welchem der Zeitverlauf analysiert werden muss. Ein Fehler wird zu einem Startzeitpunkt berichtet und ist dann über eine Zeitdauer

[11] Für eine Einführung in die Ereignisdatenanalyse siehe beispielsweise Blossfeld & Rohwer (1995), Diekmann & Mitter (1984) oder Hosmer et al. (1999).

offen, bevor er zu einem zweiten Zeitpunkt behoben wird. Im hier verwendeten Modell ist für die Fehlerberichte nur ein Zustandswechsel möglich. Sie können vom offenen Zustand in den behobenen Zustand wechseln. Dieser Zustandswechsel kann dabei jederzeit eintreten. Die Ereignisdatenanalyse versucht den dem Zustandswechsel zugrundeliegenden Prozess zu analysieren und zu erklären, welche Variablen ihn beeinflussen. Ich werde im Folgenden kurz die wichtigsten in meiner Arbeit verwendeten Begriffe der Ereignisdatenanalyse vorstellen. Auf eine detailliertere Darstellung wird jedoch verzichtet.

Zensierung Zensiert heissen Fehler, welche zum Zeitpunkt, als die Datenerhebung abgeschlossen wurde, noch nicht behoben waren. Diese Fehler müssen beispielsweise bei der Berechnung der Medianzeit berücksichtigt werden, da diese andernfalls unterschätzt würde. Wichtig für die Verfahren der Ereignisdatenanalyse ist, dass der Zensierungszeitpunkt unabhängig von den Kovariaten ist. Da die Datenerhebung zu einem zufällig gewählten Zeitpunkt beendet wurde, ist dies im vorliegenden Datensatz der Fall.

Hazardrate Die Hazardrate $r(t) = \lim_{\Delta t \to 0} \frac{q(t,t+\Delta t)}{\Delta t}$, auch Übergangsrate genannt, kann als momentane Neigung zum Zustandswechsel aufgefasst werden. Sie ist für kleine Zeitintervalle proportional zur Wahrscheinlichkeit des Zustandswechsels $q(t, t + \Delta t)$ innerhalb des Zeitintervalls $[t, t + \Delta t]$.

Überlebensfunktion Die Überlebensfunktion $G(t) = Pr(T \leq t)$ gibt die Wahrscheinlichkeit an, dass ein Fehler den Zeitpunkt t erlebt. Das heisst, dass er zu diesem Zeitpunkt noch nicht behoben wurde. Die Überlebensfunktion ist monoton fallend. Der Graph der Überlebensfunktion heisst **Überlebenskurve**.

Medianzeit Die Medianzeit ist die Zeit, zu der die Hälfte der Fehler behoben ist. Für die Medianzeit $\hat{t}$ gilt also $G(\hat{t}) = 0.5$.

Kaplan-Meier Schätzer Der Kaplan-Meier Schätzer, auch Product-Limit-Schätzer genannt, dient dazu, die Überlebensfunktion aus empirischen Daten zu schätzen. Die geschätzte Überlebensfunktion kann graphisch als Überlebenskurve dargestellt werden. Aus ihr können auch Schätzer für die Medianzeit errechnet werden.

Cox-Regression

Zur Schätzung von Kovariateneinflüssen in der Ereignisdatenanalyse gibt es verschiedene Regressionsverfahren. Grob lassen sie sich in semi-parametrische Verfahren, welche keine Annahmen zur Form der Hazardrate machen und parametrische Verfahren, bei welchen die funktionale Form der Hazardrate bestimmt ist, unterteilen. In der vorliegenden Arbeit wird, abgesehen von einer ergänzenden Regressionsrechnung im Anhang, nur das in Formel (4.1) dargestellte Verfahren der Cox-Regression verwendet (Cox 1972).

$$r(t) = h(t) \cdot e^{\beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k} = h(t) \cdot (e^{\beta_1})^{x_1} \cdot (e^{\beta_2})^{x_2} \cdot ... \cdot (e^{\beta_k})^{x_k} \tag{4.1}$$

Diese schätzt proportionale Effekte auf die Hazardrate $r(t)$ ausgehend von einer nicht spezifizierten Basis-Hazardrate $h(t)$. Bei der Cox-Regression handelt es sich deshalb um

ein semi-parametrisches Verfahren. Dies hat gegenüber einem parametrischen Modell den Vorteil, dass keine Annahmen über die Basis-Hazardrate gemacht werden müssen. Es wird einzig davon ausgegangen, dass die proportionalen Einflüsse der Variablen auf die Hazardrate im Zeitverlauf gleich bleiben.

Die Cox-Regression schätzt die Parameter β_i. Diese können jedoch nur schwer direkt interpretiert werden. Die Interpretation von e^{β_i} ist wesentlich einfacher. Deshalb wird in der Darstellung der Ergebnisse jeweils e^{β_i} angegeben. Die Hazardrate verändert sich im Cox-Modell um e^{β_i}, wenn sich die unabhängige Variable x_i um den Betrag 1 verändert. Ist $e^{\beta_i} > 1$, erhöht sich die Hazardrate und damit die Neigung zum Zustandswechsel. Falls $e^{\beta_i} < 1$ ist, dann verringert sich die Hazardrate und die Neigung zum Zustandswechsel. $100 \cdot (e^{\beta_i} - 1)$ ist die prozentuale Veränderung des durch die Hazardrate geschätzten Risikos zum Zustandswechsel, wenn sich die Variable x_i um 1 verändert. Die e^{β_i}-Effekte wirken jedoch nicht additiv, sondern multiplikativ. Eine Erhöhung der Variable x_i um 2 bewirkt deshalb nicht die doppelte prozentuale Veränderung.

Da die einzelnen Fehlerberichte des gleichen Fehlerberichterstatters nicht als voneinander unabhängig betrachtet werden können, wird ein Clustering für die Fehlerberichterstatter durchgeführt. Ohne dieses Clustering wären die Konfidenzintervalle der einzelnen Schätzer unterschätzt worden.

Die Cox-Regression liefert jedoch nur zuverlässige Schätzer, wenn die Einflüsse auf die Basis-Hazardrate im Zeitverlauf gleich bleiben. Um die Zuverlässigkeit des Modells abzuschätzen, muss diese Proportionalitätshypothese überprüft werden. Grambsch & Therneau (1994) haben dazu eine Methode vorgeschlagen, in welcher die skalierten Schoenfeld Residuen gegen die Zeit graphisch dargestellt werden. Die Schoenfeld Residuen (siehe dazu Schoenfeld (1982)) messen die Differenz zwischen dem tatsächlichen Wert einer Kovariaten und ihrem erwarteten Wert für jeden Zeitpunkt, in dem ein Ereignis stattfindet. Mit einer geglätteten Kurve der Schoenfeld Residuen kann die Zeitabhängigkeit der Kovariateneffekte β_i je einzeln graphisch dargestellt werden. Diese Kurve stellt für jede Kovariate deren Einfluss über die Zeit dar. Eine horizontale Linie bedeutet dabei, dass die Kovariateneinflüsse proportional sind.

4.4.2 Generalisierte lineare Modelle

Generalisierte lineare Modelle sind eine Verallgemeinerung der klassischen linearen Modelle. Lineare Modelle basieren auf der Annahme, dass die Realisierungen der abhängigen Variable normalverteilt sind. In generalisierten linearen Modellen kann diese Verteilung auch andere Formen annehmen. Allgemein haben generalisierte lineare Modelle folgende Form:

$$\mathrm{E}(\mathbf{Y}|[x_1, ..., x_k]) = \mu(\beta_o, x_1\beta_1, ..., x_k\beta_k) = g^{-1}(\beta_0, x_1\beta_1, ..., x_k\beta_k) \tag{4.2}$$

$\mathrm{E}(\mathbf{Y}|[x_1, ..., x_k])$ ist dabei der erwartete Mittelwerte der abhängigen Variable $\mathbf{Y}$ unter der Bedingung, dass die unabhängigen Variablen die Werte $x_1, ..., x_k$ haben. $\beta_0, ..., \beta_k$ sind die zu schätzenden linearen Einflüsse auf die unabhängigen Variablen. g ist die «Link-Funktion». In klassischen linearen Modellen ist die «Link-Funktion» nicht vorhanden. Sie ist die Identität. Ziel eines generalisierten linearen Modells ist es, die Parameter β_i zu schätzen. Dies wird meist mit der «maximum likelihood» Methode realisiert. Die

Interpretation der geschätzten Parameter hängt entscheidend von der «Link-Funktion» ab.

Aus den gleichen Gründen wie für die Cox-Regression wurden auch die verwendeten generalisierten linearen Modelle für die Fehlerberichterstatter geclustert. Allerdings ist es mit dem verwendeten R Paket nicht möglich, direkt geclusterte Varianzen zu schätzen. Deshalb wurde zur Schätzung der Varianz der einzelnen Koeffizienten ein Bootstrap-Verfahren gewählt.

Logit-Regression

Hypothese D postuliert einen Zusammenhang zwischen der Reputation des Fehlerberichterstatters und derjenigen des Fehlerbehebenden. Die abhängige Variable ist also die Reputation des Fehlerbehebenden. Je nach der zur Messung der Reputation herangezogenen Variable aus dem Datensatz ist diese jedoch kein metrisches Mass. Um ein möglichst einfaches Modell zu haben, wurde die Reputation des Fehlerbehebenden als dichotome Variable aus der Entwicklerkategorie gebildet. Dabei wurden die Kategorien Contributor und einfacher Entwickler bzw. offizieller Entwickler und Kernentwickler jeweils zusammengenommen.

Eine klassische lineare Regression ist für die Schätzung von Einflüssen auf eine dichotome Variable ungeeignet. Die lineare Regression basiert auf der Annahme, dass die abhängige Variable normalverteilt ist. Eine dichotome Variable kann aber nicht normalverteilt sein. Deshalb wird eine Logit-Regression verwendet.[12] Die Logit-Regression ist ein generalisiertes lineares Modell. Die abhängige dichotome Variable ist binomialverteilt. Die Binomialverteilung leitet sich aus der Wiederholung eines Experiments mit zwei Ausgängen ab. Entweder tritt ein Ereignis ein oder nicht. Sie eignet sich damit zur Beschreibung der Verteilung einer dichotomen Variable. Der Parameter

$$\pi([x_1, ..., x_k]) = Pr(y = 1|[x_1, ..., x_k]) \tag{4.3}$$

der Binomialverteilung wird durch die Logit-Regression geschätzt. Dieser Parameter ist die Wahrscheinlichkeit, dass das Ereignis eintritt, unter der Bedingung, dass die Kovariaten die Werte $[x_1, ..., x_k]$ haben. Übertragen auf die konkrete Anwendung ist dies die bedingte Wahrscheinlichkeit, dass ein Fehler von einem offiziellen Entwickler oder einen Kernentwickler behoben wird.

Als «Link-Funktion» wird die Logit-Funktion verwendet:

$$g(\pi) = ln(\frac{\pi}{1-\pi}) = \beta_0 + x_1\beta_1 + ... + x_k\beta_k \tag{4.4}$$

Die Umkehrfunktion g^{-1} zur Logit-Funktion macht den Zusammenhang zur Eintretens-

[12] Für eine ausführliche Darstellung der Methoden zur Analyse kategorialer Daten siehe beispielsweise Tutz (2000). Allerdings wird dort das Logit-Modell ohne Verweis auf die generalisierten linearen Modell vorgestellt. Eine kurze, prägnante Darstellung des Logit-Modells als generalisiertes lineares Modell findet sich in Long (1997: 257).

wahrscheinlichkeit deutlich:

$$\pi([x_1, ..., x_k]) = g^{-1}(\beta_0 + x_1\beta_1 + ... + x_k\beta_k) = \frac{e^{\beta_0+x_1\beta_1+...+x_k\beta_k}}{1 + e^{\beta_0+x_1\beta_1+...+x_k\beta_k}} \tag{4.5}$$

Der Term $\frac{\pi}{1-\pi}$ aus Formel (4.3) ist die Chance (engl.: odd), dass ein Ereignis eintritt im Verhältnis dazu, dass kein Ereignis eintritt. Oder auf den vorliegenden Fall angewandt, die Chance, dass ein Fehler von einem offiziellen Entwickler oder einem Kernentwickler behoben wird. Falls ein Fehler mit einer Wahrscheinlichkeit von 0.75 von einem offiziellen Entwickler oder Kernentwickler behoben wird, dann ist die Gegenwahrscheinlichkeit, dass er von einem einfachen Entwickler oder Contributor behoben wird 0.25. Die Chancen stehen demnach 3 zu 1 und es gilt $\frac{\pi}{1-\pi} = 3$.

Die Parameter β_i werden jedoch durch die «Link-Funktion» nicht als Einflüsse auf die Chancen, sondern als Einflüsse auf die logarithmierten Chance modelliert. Diese werden auch «Log-Odds» genannt. Die Parameter β_i können analog zur Interpretation für die Cox-Regression als multiplikative Einflüsse auf die Chancen verstanden werden. Aus den gleichen Gründen ist es einfacher, anstelle von β_i den Wert von e^{β_i} zu interpretieren. Ist beispielsweise $e^{\beta_i} = 1.25$, dann bedeutet dies, dass die Chance der Fehlerbehebung durch einen offiziellen Entwickler oder einen Kernentwickler um 25% steigt, falls sich der Wert der dazugehörenden Kovariaten x_i um 1 erhöht.

Poisson-Regression

Hypothese C4 postuliert einen Zusammenhang zwischen der Reputation und der Anzahl Personen, welche einen Fehlerbericht beachten. Die abhängige Variable, in diesem Fall die Anzahl Personen, ist eine diskrete Zählvariable. Es wird die Anzahl der beteiligten Personen gezählt. Der Wert dieser Variable kann also nur positive ganze Zahlen annehmen. Eine simple lineare Regression ist zur Schätzung einer solchen Variablen nur bedingt geeignet, da sie voraussetzt, dass die abhängige Variable normalverteilt ist. Insbesondere wenn, wie im vorliegenden Fall, die Werte der abhängigen Variable relativ klein sind, sind Zählvariablen jedoch kaum normalverteilt. Zur Schätzung der Anzahl Personen wird deshalb eine Poisson-Regression verwendet.[13]

Die Poisson-Regression ist eine generalisierte lineare Regression mit der in Gleichung (4.6) dargestellten Poisson-Verteilung als Verteilung der abhängigen Variable. Die Poisson-Verteilung eignet sich als Modell für die Verteilung der Anzahl eines selten auftretenden Ereignisses. Die Tatsache, dass sich eine weitere Person um einen Fehlerbericht kümmert, ist ein solches seltenes Ereignis. Die Gesamtzahl der Personen, die sich um einen Fehlerbericht kümmern ist jeweils klein. Der einzige Parameter μ dieser Verteilung ist der Mittelwert.

$$P(Y = y|\mathbf{x}) = \frac{e^{-\mu}\mu^y}{y!} \tag{4.6}$$

[13] Cameron & Trivedi (1998) enthält eine detaillierte Darstellung der Poisson-Regression.

In der Poisson-Regression wird nun der Einfluss der unabhängigen Variablen auf den Parameter μ der Verteilung geschätzt. Damit $\mu > 0$ gilt, werden die Einflüsse nicht linear, sondern ähnlich wie in der Cox-Regression mit folgender «Link-Funktion» modelliert:

$$\mu = e^{\beta_0 + x_1\beta_1 + ... + x_k\beta_k} \tag{4.7}$$

Die geschätzten Koeffizienten einer Poisson-Regression können deshalb gleich wie die Koeffizienten der Cox-Regression interpretiert werden. Es handelt sich um multiplikative Effekte auf die Chance, dass sich eine zusätzliche Person an einem Fehlerbericht beteiligt, wenn die zugehörige unabhängige um 1 grösser wird.

Da die Poisson-Distribution nur einen einzigen Parameter hat, kann sich die Varianz nicht unabhängig vom Mittelwert ändern. Falls die Varianz grösser als erwartet ist, dann spricht man von «Overdispersion». Zur Prüfung der Validität des Poisson-Modells müsste auch geprüft werden, ob keine «Overdispersion» vorliegt. Da sich jedoch im empirischen Test gezeigt hat, dass der vermutete Zusammenhang nicht vorhanden ist, wurde auf die Prüfung der «Overdispersion» verzichtet.

5 Empirische Ergebnisse

Falls nicht anders angegeben beziehen sich die Daten im Folgenden nur auf den bereinigten Datensatz. Die Daten aus dem unbereinigten Teil des Datensatzes können nur dort verwendet werden, wo die korrekte Zuordnung zu einer bestimmten Person keinen Rolle spielt.

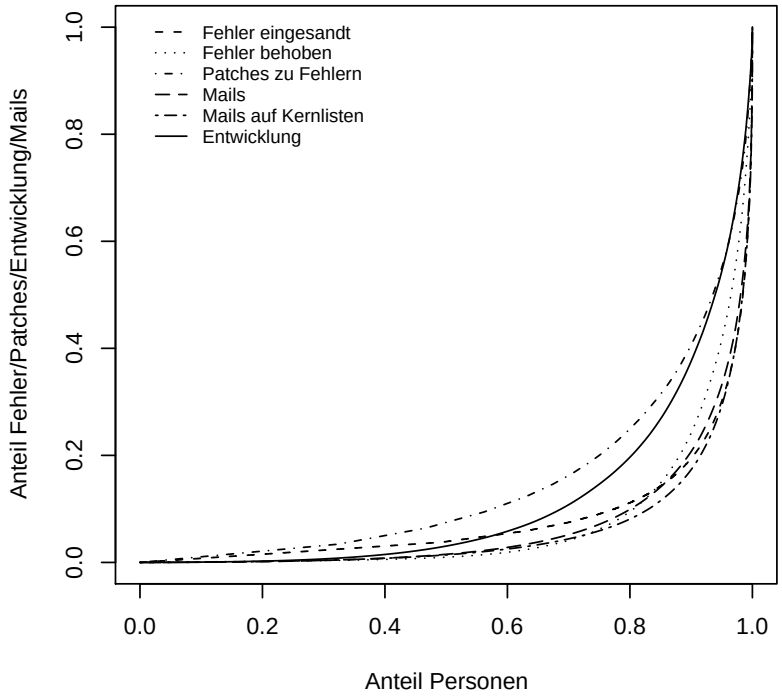
5.1 Deskriptive Ergebnisse

5.1.1 Verteilung der Arbeit

Dass die Arbeit im Debian Projekt sehr ungleich auf die Personen verteilt ist, zeigen bereits die Gini-Koeffizienten in Tabelle 4.4. Die in Abbildung 5.1 dargestellten Lorenzkurven verdeutlichen diese Tatsache noch graphisch.

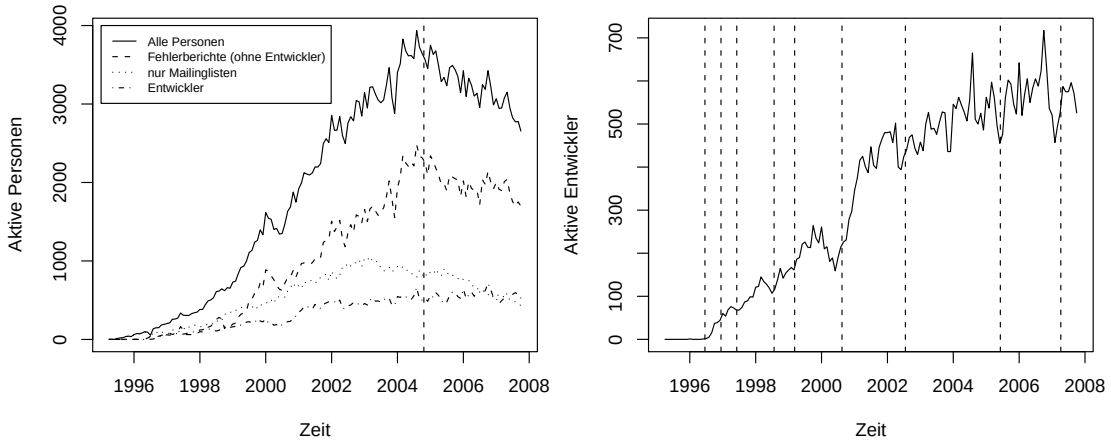
Diese sehr schräge Verteilung der Arbeit korrespondiert mit den Erkenntnissen von Ghosh & Ved Prakash (2000). Diese haben bei ihrer Analyse des Codes zahlreicher Open Source Projekte herausgefunden, dass sich die 10% aktivsten Entwickler für 72.3% des Codes verantwortlich zeichnen. Alleine die 10 aktivsten Entwickler haben zusammen 19.8% des Codes programmiert. Eine ausführlichere und genauere Analyse der Verteilung der Arbeit im GNOME Projekt wurde durch Koch & Schneider (2000) durchgeführt. Sie kommen zum Schluss, dass die Anzahl der programmierten Codezeilen sehr ungleich verteilt ist. Ihr Datensatz umfasst jedoch nur knapp 300 Programmierer. Interessant in ihrer Untersuchung ist insbesondere die Tatsache, dass die Zeit, welche die einzelnen Programmierer bereits im Projekt aktiv waren viel weniger schief verteilt ist. Die ungleiche Verteilung der Arbeitsleistung ist also nicht darauf zurückzuführen, dass nur sehr wenige Personen sehr lang im Projekt aktiv sind. Eher anekdotisch wertvoll ist hingegen die in Lerner & Tirole (2005: 54) zitierte Aussage aus einem internen Dokument von Microsoft. Dieses Dokument wurde der Open Source Initiative zugespielt und erhielt unter dem Titel «Halloween Document» einige Berühmtheit. In diesem wird vor der Gefahr von Open Source Software für das Unternehmen gewarnt. Dabei wird unter anderem das Argument verbreitet, dass zu jedem Entwickler, der Code beitrage, fünf weitere kommen, welche nur Fehler berichten. Im Debian Projekt ist dieses Verhältnis sogar noch extremer. Auf jeden Entwickler kommen über 11 Personen, welche nur Fehler berichten. Lerner & Tirole bemerken jedoch richtigerweise, dass auch die Produktivität der einzelnen Programmierer in kommerziellen Softwareunternehmen sehr ungleich verteilt ist. Ob sich diese Verteilungen gleichen und auf die gleichen Ursachen zurückzuführen sind, müsste weiter untersucht werden.

Abbildung 5.1: Lorenzkurven der Verteilung der Fehler, Mails und Arbeit pro Entwickler



Anmerkung: Lorenzkurven der totalen Aktivität über den ganzen Untersuchungszeitraum für jede Person; N (Anzahl Personen): 32'253 für Fehler eingesandt, 3'850 für Fehler behoben, 1'313 für Patches, 12'908 für Mails, 3'935 für Mails auf Kernlisten, 2'512 für Entwicklung, Es wurden jeweils nur diejenigen Personen berücksichtigt, welche in der entsprechenden Aktivität einen Wert > 0 aufweisen; Für die Mailinglisten wurden nur Personen im bereinigten Datensatz berücksichtigt.

Abbildung 5.2: Zeitliche Entwicklung des Debian Projekts



Anmerkung: N = 32'539 Personen; Anzahl der aktiven Personen nach Tätigkeit im jeweiligen Monat. Die vertikale Linie in der linken Abbildung markiert das erste Release der Distribution Ubuntu. Die Linien in der rechten Abbildung markieren die Releases der Debian Distribution. Rechts sind nur die aktiven Entwickler geplottet. Die Werte für «nur Mailinglisten» umfassen nur Personen im bereinigten Datensatz. Personen, welche nie einen Fehler berichtet haben sind nicht eingeschlossen. Die Gesamtzahl der auf den Mailinglisten aktiven Personen ist wesentlich grösser.

5.1.2 Zeitliche Entwicklung des Debian Projekts

Ich möchte im Folgenden kurz auf die zeitliche Entwicklung des Debian Projekts eingehen. Das Projekt wurde im August 1993 ins Leben gerufen. Allerdings sind im Datensatz für die ersten Jahre keine Daten vorhanden. Der Datensatz startet deshalb erst im April 1995. Zu diesem Zeitpunkt war das Projekt aber im Vergleich zu heute immer noch sehr klein.

Abbildung 5.2 zeigt die zeitliche Entwicklung des Debian Projekts. Auf der y-Achse ist die Anzahl der Personen abgetragen, welche im jeweiligen Monat aktiv sind. Das Diagramm zeigt, dass die Distribution bis ins Jahr 2004 gewachsen ist. Das Wachstum verläuft zwar nicht ganz kontinuierlich. Es gibt immer wieder Phasen mit markant reduzierter Aktivität. Der Trend ist aber klar steigend. Ab 2004 ist ein Rückgang bemerkbar. Die vertikale Linie im linken Diagramm markiert das erste Release der Linux Distribution Ubuntu, welche auf Debian basiert und insbesondere Desktopbenutzer ansprechen soll. Vermutlich ist der Rückgang der aktiven Personen auf eine Abwanderung zu Ubuntu zurückzuführen. Die Unterteilung nach den verschiedenen Tätigkeiten zeigt, dass insbesondere weniger Personen sich als Fehlerberichterstatter betätigen. Die Anzahl der Entwickler nimmt dagegen nicht ab. Daraus lässt sich schliessen, dass insbesondere Benutzer zu Ubuntu abgewandert sind, jedoch nur wenige Entwickler.

Die Diskontinuitäten im Wachstum sind wahrscheinlich zufällige Schwankungen und auf Ereignisse innerhalb des Projekts zurückzuführen. Auffällig ist der Zusammenhang zum Releasezyklus. Das rechte Diagramm in Abbildung 5.2 zeigt die per Monat aktiven Entwickler. Dieses Diagramm ist eine Vergrösserung der untersten (nur Entwickler) Linie im linken Diagramm. Die vertikalen Linien im Diagramm markieren die Releases der

Debian Distribution. Auffällig ist, dass jeweils vor jedem Release die Aktivität zuerst ein lokales Hoch und danach kurz vor dem Release ein lokales Tief erreicht. Dies hat damit zu tun, dass auf ein Release hin die Arbeit intensiviert wird und ganz kurz vor dem Release die Distribution eingefroren wird und nur noch wichtige Fehler behoben werden können. Das Diagramm zeigt auch, dass sich das Wachstum der Zahl der Entwickler ab ca. 2004 verlangsamt.

Die zeitliche Entwicklung der Anzahl der Fehler und der Arbeit an Paketen verläuft nahezu parallel zu der Anzahl der Personen, die im jeweiligen Bereich arbeiten.

5.1.3 Regionale Verteilung

Der Datensatz enthält für Emailnachrichten, Fehlerberichte und hochgeladene Softwarepakete auch die Zeitzone, in der die jeweilige Arbeit geleistet wurde. Diese Information stammt aus der Konfiguration des Computers des Entwicklers, welche automatisch beim versenden einer Nachricht oder beim erstellen eines Softwarepakets als Teil des Zeitstempels eingefügt wird. Interessant ist die Zeitzone deshalb, weil sich aus ihr Rückschlüsse auf den Aufenthaltsort eines Entwicklers machen lassen. Allerdings können die Entwickler so nur in der geographischen Länge verortet werden. So ist es möglich, dass ein Teil der Westeuropa zugeordneten Beiträge aus Afrika stammen. Aus dem gleichen Grund, kann auch nicht zwischen Nord- und Südamerika unterschieden werden. Die in Abschnitt 2.1.1 eingehender vorgestellten FLOSS Surveys zeigen jedoch übereinstimmend, dass es in Afrika praktisch keine Entwickler gibt und dass der überwiegende Teil der Entwickler des amerikanischen Kontinents im Norden lebt. Die Daten für die entsprechenden Zeitzonen können deshalb als gute Näherung für die in Nordamerika bzw. Westeuropa geleistete Arbeit betrachtet werden.

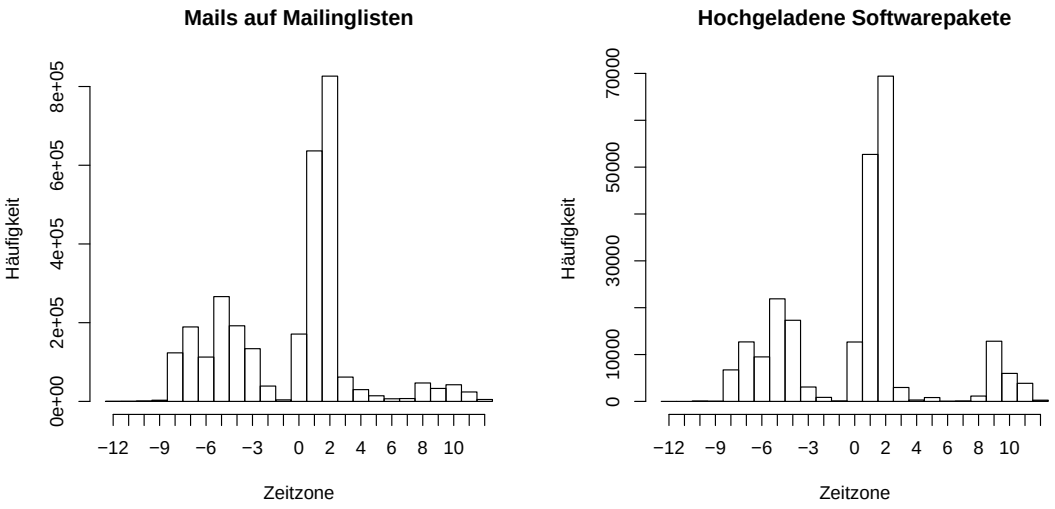
Abbildung 5.3 zeigt die Verteilung der Arbeit an Softwarepaketen und der Mails auf Mailinglisten auf die verschiedenen Zeitzonen. Die Darstellung bezieht sich auf den Anteil der Arbeit (geschriebene Emailnachrichten bzw. hochgeladene Pakete), welche in der entsprechenden Zeitzone geleistet wurde und nicht auf die Anzahl der Personen, welche in der entsprechenden Zone leben. Eine Zuordnung auf einzelne Personen wäre wesentlich aufwändiger, da etliche Personen im Verlaufe der Untersuchung in mehreren Zeitzonen gelebt haben.

Tabelle 5.1: Verteilung der Arbeit auf Weltregionen

Region	**Zeitzonen**	**Mailinglisten**	**Softwarepakete**	**POLS**	**US**
Amerika	-8 – -2	34%	30%	14%	27%
Westeuropa	0 – 2	55%	57%	70%	53%
Australien & Japan	8 – 11	5%	10%		

Anmerkung: N: 235'529 hochgeladene Pakete, 2'973'071 Mails von Mailinglisten; Zeitzone = Abweichung der aktuellen Zeit von GMT, keine Korrektur für Sommer- bzw. Winterzeit; POLS und US bezieht sich auf die Ergebnisse der FOSS Surveys aus Tabelle 2.1.

Abbildung 5.3: Verteilung der Arbeit über die Zeitzonen



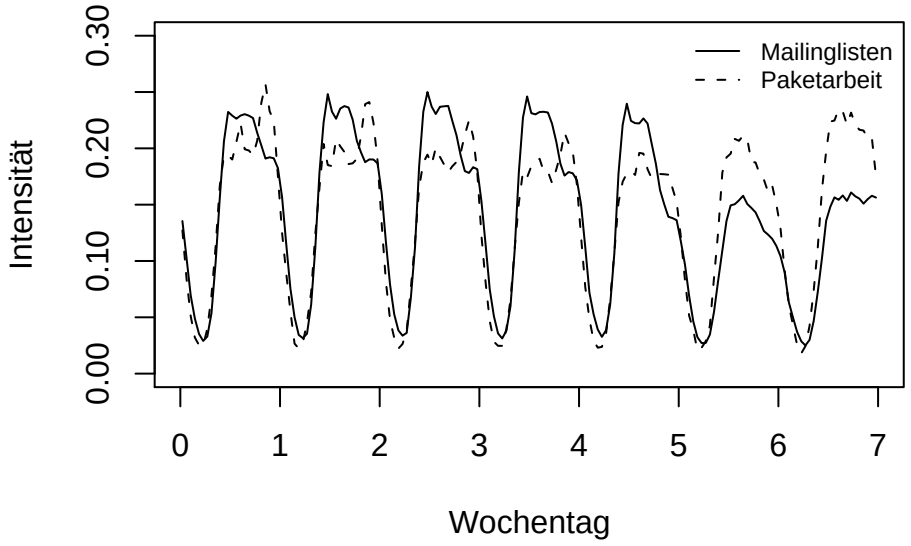
Anmerkung: N: 235'529 hochgeladene Pakete, 2'973'071 Mails von Mailinglisten; Zeitzone = Abweichung der aktuellen Zeit von GMT, keine Korrektur für Sommer- bzw. Winterzeit

Anschaulicher werden die Ergebnisse, wenn man sie auf die Weltregionen zusammenfasst (Tabelle 5.1). Die im Histogramm sichtbaren Unterschiede innerhalb dieser Regionen sind mir Vorsicht zu geniessen, da die Zeitzone nicht für die Sommer- bzw. Winterzeit korrigiert wurde. Der gleiche geographische Ort ist also je nach Jahreszeit in einer anderen Zeitzone. Die Verortung ist deshalb relativ grob. Für eine Zusammenfassung nach Weltregionen reicht dies jedoch aus. Interessant ist auch ein Vergleich mit den Ergebnissen der grossen FLOSS Surveys. Diese Zusammenfassung bestätigt im Grossen und Ganzen die Befunde der Befragungen, dass ein Grossteil der Entwicklung von freier Software in Amerika und Westeuropa geschieht. Die Zahlen zum Debian Projekt entsprechen dabei eher den Angaben zum aktuellen Wohnsitz aus der FLOSS-US (David et al. 2003) Befragung. Das Verhältnis zwischen der Arbeit in Westeuropa und in Nordamerika entspricht ungefähr dem Verhältnis der Populationen. Der Anteil der Paketarbeit ist in Westeuropa etwas höher als in Nordamerika. Dies deutet darauf hin, dass ein etwas überproportionaler Anteil der Entwickler im Unterschied zu den restlichen im Projekt aktiven Personen (Mailinglisten, Fehlerberichte) in Europa lebt.

5.1.4 Bezahlte Arbeit und Freiwilligenarbeit

Über die Verteilung der Arbeit auf die verschiedenen Tageszeiten und Wochentage können Rückschlüsse darauf gezogen werden, ob der Beitrag als Teil einer bezahlten Tätigkeit oder als Freiwilligenarbeit geleistet wurde. Abbildung 5.4 zeigt, dass ein grosser Anteil der Arbeit während Arbeitszeiten geleistet wird. Allerdings ist der Anteil der während der Arbeitszeit versandten Mails wesentlich grösser als der Anteil der Paketarbeit. Die

Abbildung 5.4: Verteilung der Arbeit über die Wochentage und Tageszeiten



Anmerkung: N: 235'529 hochgeladene Pakete, 2'973'071 Mails von Mailinglisten; Histogramm der relativen Häufigkeit mit einer Klassenbreite von 1 Stunde; Der Nullpunkt entspricht Montagmorgen um 0 Uhr.

Paketarbeit zeigt ihre höchste Intensität in den Abendstunden und ist am Wochenende nicht wesentlich weniger intensiv als unter der Woche, jedoch ohne den Peak in den Abendstunden. Dies deutet darauf hin, dass doch ein grosser Teil der Entwicklungsarbeit in Freiwilligenarbeit geleistet wird. Die stärkere Konzentration der Mailinglistenaktivität auf die Arbeitsstunden lässt sich auch dadurch erklären, dass darin zu einem grossen Teil auch Mails von Anwendern enthalten sind, die Debian bei ihrer Arbeit einsetzen. Allerdings wird auch ein nicht geringer Teil der Paketarbeit während regulärer Arbeitsstunden geleistet und es ist davon auszugehen, dass ein grosser Teil dieser Arbeit von Personen geleistet wird, welche für diese Arbeit direkt oder indirekt bezahlt werden. Diese graphische Interpretation deckt sich mit den Antworten auf die Frage nach der monetären Entschädigung (siehe Abschnitt 2.1.1) für die Arbeit an freier Software aus der FLOSS-POLS Studie (Ghosh et al. 2002). Dort geben 46% an, keinerlei Entschädigung zu erhalten. Von denjenigen, die eine monetäre Entschädigung in irgendeiner Form erhalten, sind nur ca. 30% für die Entwicklung von freier Software am Arbeitsplatz bezahlt. Zu ähnlichen Ergebnissen kommen auch Hars & Ou (2002) in ihrer wesentlich kleineren Befragung.

Die Darstellung in Abbildung 5.4 lässt sicherlich keine sehr genauen Schlüsse über die Verteilung von bezahlter und Freiwilligenarbeit zu. Sie bestätigt jedoch für das Debian Projekt die Erkenntnisse aus Befragungen von Open Source Entwicklern, dass die Vorstellung, dass Open Source Software nur von Freiwilligen in ihrer Freizeit entwickelt wird, falsch ist. Aus der Tatsache, dass Open Source Software zu einem grossen Teil auch während der normalen Arbeitsstunden entwickelt wird, lassen sich nicht direkt Rückschlüsse darauf ziehen, ob und in welchem Umfang die entsprechenden Entwickler dafür bezahlt werden. Die Vermutung liegt jedoch nahe, dass sie diesen Zeitaufwand zu

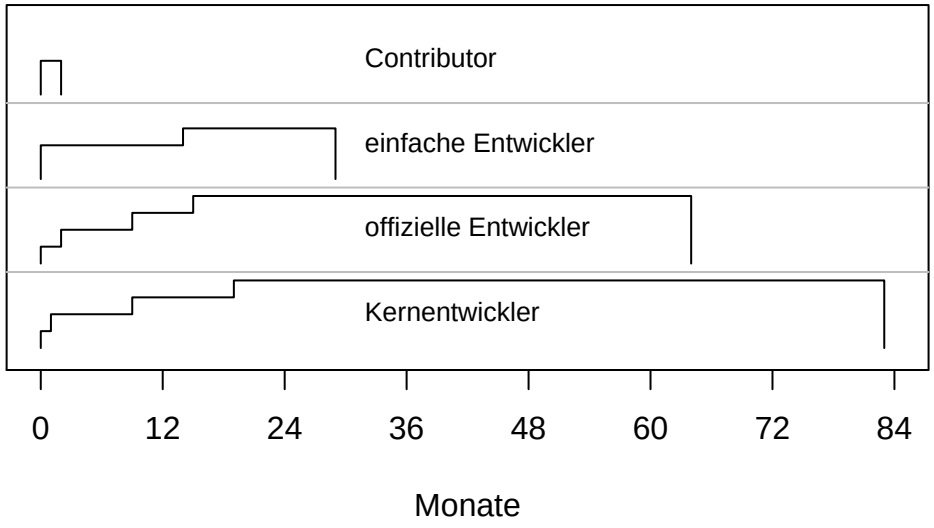
diesen Tageszeiten kaum gänzlich ohne monetäre Entschädigung leisten können.

5.2 Entwicklerkarrieren

Das Engagement für das Debian Projekt verläuft bei den meisten Entwicklern über mehrere Stufen. Am häufigsten ist, dass sich eine Person zuerst auf den Mailinglisten beteiligt, danach Fehler berichtet, dann aktiv an Paketen mitarbeitet und zuletzt offizieller Entwickler wird. Abbildung 5.5 fasst diesen Karriereverlauf für die verschiedenen Entwicklerkategorien zusammen.

Die Darstellung zeigt ein stark generalisiertes Bild. Neben dieser Modellkarriere kommen auch andere Verläufe vor. So beteiligen sich ca. 20% der Personen von Begin weg aktiv als Entwickler am Projekt. Auch geht die Darstellung davon aus, dass eine Person, welche eine Aktivitätsstufe erreicht hat, danach nicht mehr auf eine tiefere Stufe zurückfällt. In Realität kann aber auf eine Periode, in der ein Entwickler aktiv mitarbeitet, auch wieder eine Periode folgen, in der er sich nur an Diskussionen auf den Mailinglisten und mit Fehlerberichten beteiligt. Der Übergang zur Kategorie der Kernentwickler kann aufgrund der Definition der Kernentwickler in dieser Form nicht dargestellt werden, da die Kernentwickler dadurch definiert sind, dass sie global zu den 20% aktivsten Entwicklern gehören. Wann aber genau der Übergang in diese Gruppe erfolgt, ist aus den Daten nicht

Abbildung 5.5: Zeitlicher Verlauf der «Entwicklerkarrieren» nach Status



Anmerkung: Die x-Achse ist die Zeit seit Beginn der Beteiligung am Projekt. Die Treppen markieren folgende Karrierestufen: nur Mailinglistenaktivität, Contributor, einfacher Entwickler, offizieller Entwickler. Die Unterteilung erfolgt nach dem höchsten erreichten Status. Übergangszeitpunkt in die nächsthöhere Stufe ist jeweils die Mediandauer bis zu diesem Übergang ab Beginn der Projektbeteiligung. Anzahl (N): Contributor 30'027, einfache Entwickler 1'181, offizielle Entwickler 828, Kernentwickler 503

zu ermitteln.

Die Dauer der Beteiligung am Projekt ist grob invers proportional zum Umfang der Beteiligung. Die mit Abstand grösste Gruppe der reinen Contributoren beteiligt sich im Median nur sehr kurz am Projekt, während die aktiv Mitarbeitenden sich wesentlich länger beteiligen. Sie bleiben auch länger auf der Stufe des Contributors als diejenigen, die sich nie aktiv an der Entwicklung beteiligen werden. Beim Übergang zu den offiziellen Entwicklern verhält es sich umgekehrt. Diejenigen, welche offizielle Entwickler werden, erreichen diesen Status vergleichsweise kurze Zeit nachdem sie begonnen haben, sich aktiv zu beteiligen. Die Darstellung zeigt, dass sich die reinen Contributoren nur sehr kurz am Projekt beteiligen. Die Periode während der sich eine Person nur auf Mailinglisten beteiligt ist sehr kurz. Bei den Contributoren und den einfachen Entwicklern ist sie im Median sogar 0. Diese bedeutet, dass mehr als die Hälfte der Personen, in diesen Kategorien sich entweder gar nicht auf Mailinglisten beteiligen oder dies zum ersten Mal im gleichen Monat tun, in dem sie auch ihren ersten Fehler berichten. Da die Daten zur Beteiligung an Mailinglisten jedoch nicht bereinigt sind, kann es sein, dass teilweise Beiträge auf Mailinglisten nicht mitgezählt werden.

Die Intensität der Aktivität ist bei den meisten Personen über den Zeitraum ihrer Beteiligung jedoch relativ gering. So sind die hier als durchgehend dargestellten Phasen immer wieder von Perioden der Inaktivität unterbrochen. Tabelle 5.2 gibt einen Überblick über die Aktivitätsperioden.

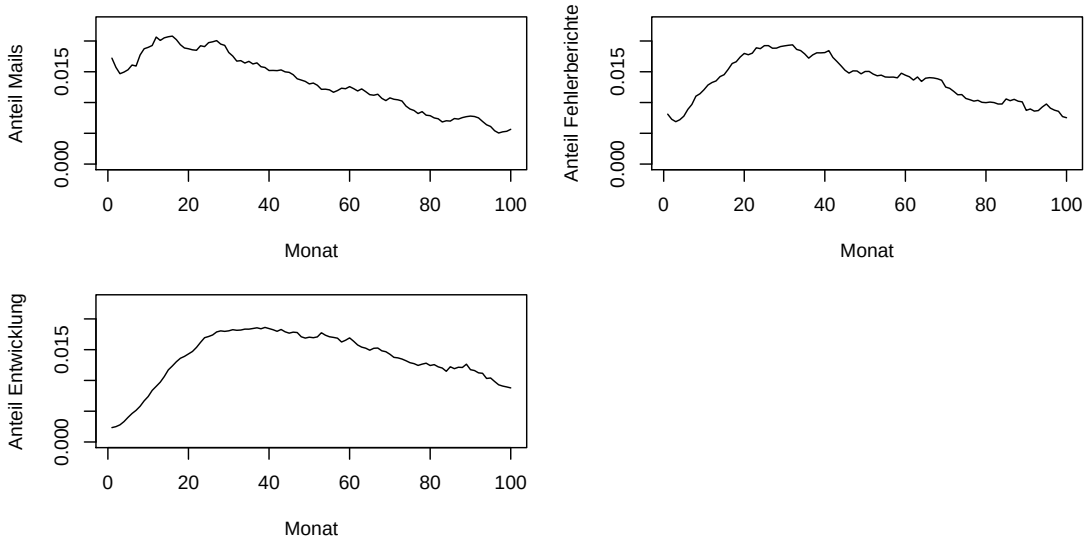
Im Mittel ist eine Person nur in 65% der Monate während ihrer Beteiligung am Projekt auch wirklich aktiv. Die Aktivität verteilt sich dabei im Schnitt auf 3.57 Perioden. Die Dauer der Perioden wurde mit dem Kaplan-Meier Verfahren geschätzt. Je stärker das Engagement für das Projekt, desto länger werden die einzelnen Perioden und umso kürzer werden die Perioden der Inaktivität dazwischen. Auch die Anzahl der Perioden nimmt mit dem Engagement zu, wobei allerdings die Kernentwickler eine geringer Anzahl aufweisen als die restlichen Entwickler. Der Beitrag der Kernentwickler ist also nicht nur

Tabelle 5.2: Aktivitätsperioden nach Status

	Anz. Perioden	**Mediandauer**		**Anteil Gesamtdauer**	
		Aktiv	**Inaktiv**	**Alles**	**nur Entw.**
alle Personen	3.57	1	2	65%	3%
Contributor	3.24	1	3	65%	-
einfache Entwickler	6.38	2	2	58%	34%
offizielle Entwickler	10.16	2	2	60%	44%
Kernentwickler	6.27	3	1	86%	70%

Anmerkung: Mittelwert der Anzahl Perioden und Mediandauer der aktiven und inaktiven Perioden jeweils in Monaten. Anteil an der Gesamtdauer ist der Anteil der Monate mit Aktivität an der Gesamtzahl der Monate vom ersten bis zum letzten Beitrag. In der Spalte nur Entwicklung ist nur die Aktivität als Entwickler (Arbeit an Softwarepaketen) berücksichtigt. Anzahl (N): alle Personen 32'539, Contributor 30'027, einfache Entwickler 1'181, offizielle Entwickler 828, Kernentwickler 503

Abbildung 5.6: Arbeitsintensität im Zeitverlauf: Mails, Fehlerberichte, Entwicklung



Anmerkung: x-Achse: Monat seit dem Eintritt ins Projekt, y-Achse: Mittelwert der Anteile an der gesamten Arbeit der Person, welche in diesem Monat geleistet wurde, geglättet über 5 Monate. N=461 nur Personen, welche sich über mindestens 50 Monate beteiligen, mindestens 80% davon aktiv sind und sich mindestens in 33 Monaten als Paketentwicklung beteiligen.

grösser, sondern auch konstanter. Auch weisen sie kürzere Perioden der Inaktivität auf. Diese Tatsache wird auch aus dem Anteil der Aktivitätsperioden an der Gesamtdauer (vom ersten bis zum letzten Beitrag) der Beteiligung ersichtlich. Insbesondere haben die Kernentwickler einen wesentlich höheren Anteil an Monaten, in denen sie sich auch aktiv als Entwickler beteiligen.

Diese Darstellung zu den Aktivitätsperioden zeigt aus einem anderen Blickwinkel als die Darstellung der Arbeitsverteilung als Lorenzkurven in Abbildung 5.1, dass sich der grösste Teil der Entwickler relativ extensiv am Projekt beteiligt und nur ein kleiner Anteil sehr intensiv und konstant daran arbeitet. Allerdings ist diese Gruppe der Kernentwickler in absoluten Zahlen immer noch gross. Zum Zensierungszeitpunkt der Datenerhebung beinhaltet sie 424 aktive Personen. Insgesamt gehören ihr 503 Personen an.

Abbildung 5.6 zeigt die durchschnittliche Arbeitsintensität im Zeitverlauf. Mit dieser Abbildung soll die Hypothese B zur Reputationsbildung «geprüft» werden. Diese Hypothese besagt, dass Entwickler anfänglich mehr Arbeit ins Projekt investieren und danach von der durch ihre Arbeit generierten Reputation profitieren und deshalb weniger investieren.

Da, wie bereits weiter oben dargestellt, der Arbeitseinsatz in den meisten Fällen nicht kontinuierlich geschieht, sondern Phasen hoher Aktivität mit Phasen ohne oder mit niedriger Aktivität abwechseln, wurde die monatliche Arbeitsaktivität auf dem Individualniveau geglättet. Dazu wurden als Aktivität eines Monats jeweils der Durchschnitt der Aktivität des aktuellen Monats und der Aktivität der zwei vorausgehenden und der zwei folgenden

Monate genommen. Ohne diese Glättung waren die Aktivitätskurven so «unruhig», dass sich der grössere Trend nur schwer erkennen lässt. Zudem wurden für die Aktivitätskurven nur diejenigen Personen berücksichtigt, die sich über eine längere Zeit und mit einer gewissen Konstanz am Projekt beteiligten. Diese Einschränkung wurde vorgenommen, damit die Betrachtung nicht durch den grossen Anteil von Personen verfälscht wird, die sich nur einmalig oder sehr kurz an dem Projekt beteiligen. Weiter dient diese Einschränkung auch dazu, Entwickler auszuschliessen, die ihre Aktivität erst kurz vor dem Zensierungszeitpuntk begonnen haben.

Die Abbildung zeigt, dass sich Hypothese B für den eingeschränkten Kreis der langfristig aktiven Entwickler «bestätigen» lässt. Zudem zeigt die Abbildung auch eine Abfolge der verschiedenen Entwickleraktivitäten. Die Aktivität auf den Mailinglisten erreicht ihren Höhepunkt zuerst, danach die Aktivität der Fehlerberichterstattung und erst später die Aktivität in der Paketarbeit. Sowohl in der Mailinglistenaktivität wie auch in der Aktivität der Fehlerberichte lässt sich ein kleiner Abfall in den ersten Monaten beobachten. Dies lässt sich rein technisch begründen. Im ersten Monat müssen alle Personen des Datensatzes eine Aktivität aufweisen, da so der Start ihrer Entwicklerkarriere definiert ist. Die Wahrscheinlichkeit einer Aktivität im ersten Monat ist deshalb höher als in den folgenden Monaten, wo sie die verschiedenen Aktivitätsmuster von aktiven und passiven Phasen überlagern. Dass dieser Abfall bei der Paketarbeit nicht sichtbar ist, liegt darin, dass die wenigsten Entwickler sich bereits im ersten Monat in diesem Bereich betätigen. Die Kurve der Paketarbeit zeigt auch, dass hier der Anstieg am längsten dauert und danach relativ flach abfällt.

5.3 Kooperation in der Fehlerbehebung

Im Theorieteil dieser Arbeit habe ich gezeigt, dass Informationen über das Kooperationsverhalten der jeweiligen Interaktionspartner wichtig sind, um die Kooperation in einem Open Source Projekt aufrechtzuerhalten. Solche Informationen können entweder durch wiederholte Interaktion (Hypothese A) oder als Reputations- und Statussignale (Hypothesen C & D) vorliegen. Ergänzend dazu wurde ausgehend vom Freiwilligendilemma ein Effekt der Verantwortungsdiffusion postuliert. Je mehr Personen für einen bestimmten Fehlerbericht potentiell zuständig sind, desto unklarer sind die Verantwortlichkeiten. Dies führt dazu, dass die entsprechenden Fehler weniger oft und weniger schnell behoben werden (Hypothese E). Im folgenden geht es nun darum, diese im Theorieteil herausgearbeiteten Mechanismen am empirischen Material zu prüfen. Dazu dient der in Abschnitt 4.3.1 vorgestellte Datensatz zu den Fehlerberichten.

Der Grossteil der Hypothesen bezieht sich auf die Hazardrate der Fehlerbehebung und auf die Zeit bis zur Behebung eines Fehlers. Zur Prüfung dieser Hypothesen dienen Methoden der Ereignisdatenanalyse (siehe Abschnitt 4.4.1). Im folgenden werden zuerst erste deskriptive Ergebnisse zum Datensatz und dann die zentralen Cox-Regressionen zur Überlebenszeit der Fehlerberichte vorgestellt und diskutiert. Im Anschluss daran wird näher auf die ergänzenden Hypothesen zur Beachtung der Fehlerberichte (Hypothese D) und zur Reputation der Fehlerbehebenden (Hypothese C4) eingegangen. Die meisten

Analysen werden jeweils für den vollständigen und den reduzierten Datensatz durchgeführt. Wo sich wesentliche Unterschiede in den Datensätzen zeigen, versuche ich, diese genauer zu untersuchen und zu erklären. Während im vollständigen Datensatz nur Fehlerberichte ausgeschlossen wurden, welche sich überhaupt nicht für die Analyse eignen, enthält der reduzierte Datensatz nur ein Subsample der Fehlerberichte. Die Kriterien zur Bildung dieses Subsamples wurden so gewählt, dass die darin enthaltenen Fehlerberichte möglichst homogen sind. Es wurde versucht, möglichst alle Einflüsse auszuschliessen, die gar nicht oder nur ungenügend kontrolliert werden können.

5.3.1 Kaplan-Meier Schätzer und Überlebenskurven

Bevor ich mich komplizierteren Modellen zur Erklärung der Überlebenszeit zuwende, präsentiere ich in Tabelle 5.3 einige einfache Schätzungen der Medianüberlebenszeit und in Abbildung 5.7 eine graphische Darstellung der Überlebenskurven.

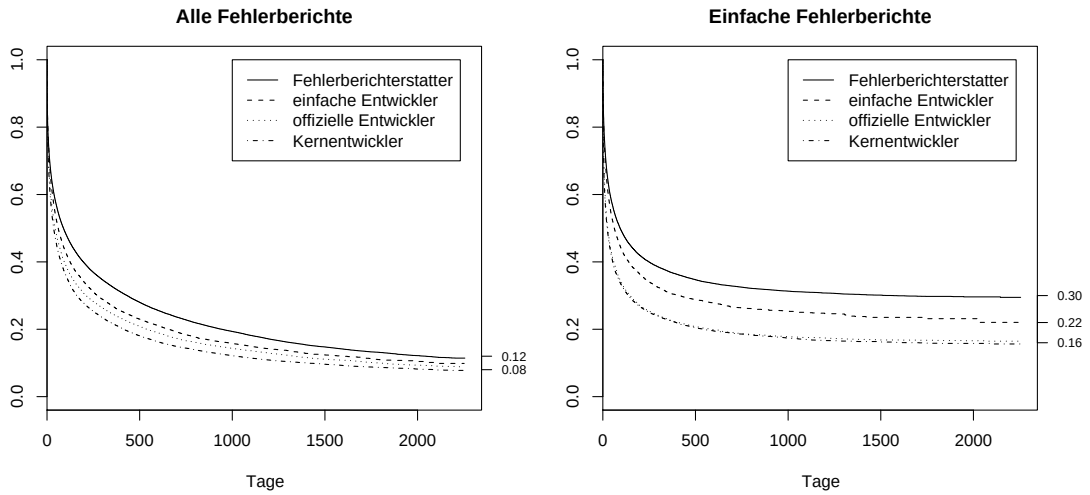
Tabelle 5.3: Kaplan-Meier Überlebenszeiten nach Entwicklerstatus

Status	**N**	**Ereignisse**	**Median**	**95% Konf.int.**
vollständiger Datensatz				
Alle	305’342	241’661	63.4	62.5 – 64.3
Contributor	170’799	132’378	88.6	86.9 – 90.4
einfacher Entwickler	31’923	23’702	60.5	58.1 – 62.6
offizieller Entwickler	33’841	28’494	44.0	42.3 – 45.7
Kernentwickler	67’231	55’844	35.8	34.9 – 36.8
reduzierter Datensatz				
Alle	131’382	89’917	53.8	52.8 – 55.1
Contributor	67’496	43’041	93.2	90.6 – 96.4
einfacher Entwickler	15’824	9’935	61.8	58.8 – 65.9
offizieller Entwickler	15’132	11’822	26.3	24.5 – 28.1
Kernentwickler	32’314	24’703	23.7	22.6 – 24.7

Anmerkung: Zu 1’548 bzw. 598 Fehlerberichten fehlt die Information zum genauen Zeitpunkt, zu dem eine Person offizieller Entwickler wurde. Diese Fehlerberichte fehlen in der Detailaufstellung. Alle Zeiten sind in Tagen.

Aus Tabelle 5.3 wird bereits ersichtlich, dass der Status des Fehlerberichterstatters sowohl im kompletten, wie auch im reduzierten Datensatz einen Einfluss auf die Medianzeit bis zur Fehlerbehebung hat. In beiden Datensätzen sind die Unterschiede signifikant. Einzig der Unterschied zwischen den offiziellen Entwicklern und den Kernentwicklern im Datensatz der einfachen Fehler ist nicht signifikant. Die Medianzeit bis zur Fehlerbehebung wird für Kernentwickler gegenüber allen Entwicklern ungefähr halbiert. Für die einfachen Fehler, bei denen die Effekte ausgeprägter sind, wird ein Fehler eines Kernentwicklers fast vier Mal schneller behoben als derjenige eines Contributors.

Abbildung 5.7: Überlebenskurven nach Entwicklerkategorie



Anmerkung: N siehe Tabelle 5.3; Die nicht dargestellten 95%-Konfidenzintervalle der Überlebenskurven zeigen, dass der Unterschied zwischen den Entwicklerkurven im vollständigen Datensatz nicht über die ganze Zeit signifikant ist. Der Unterschied zu den Contributoren ist jedoch auch am Ende knapp signifikant. Im Datensatz der einfachen Fehlerberichte sind nur die Unterschiede zwischen den offiziellen Entwicklern und den Kernentwicklern nicht signifikant.

Abbildung 5.7 bestätigt diese Erkenntnisse weiter. Im kompletten Datensatz liegen aber die Überlebenskurven der Fehler aller drei Entwicklerkategorien relativ eng beieinander und sind zum Ende auch nicht mehr signifikant. Es zeigt sich dort eine Zweiteilung zwischen den Contributoren einerseits und den Entwicklern andererseits. Diese ist auch in den Medianzeiten ersichtlich. Der Unterschied zwischen den Contributoren und den einfachen Entwicklern ist dort am grössten. Bei den einfachen Fehlern ist eine Dreiteilung erkennbar. Die Unterschiede zwischen den offiziellen Entwicklern und den Kernentwicklern verschwinden fast vollständig. Die Unterschiede zu den einfachen Entwicklern sind dagegen ausgeprägter. Dies zeigt sich auch in den Medianzeiten. Allerdings ist die Varianz bei den einfachen Entwicklern am grössten, was sich in einem grösseren Konfidenzintervall zeigt.

Die Überlebenskurven zeigen auch, dass ein gewisser Anteil der Fehler auch nach längerer Zeit nicht behoben wird. Eine horizontale Kurve bedeutet, dass in diesem Zeitraum keine Fehler behoben werden. Im kompletten Datensatz wird die Überlebenskurve erst sehr spät horizontal. Auch nach über 1000 Tagen werden immer noch Fehler behoben. Im reduzierten Datensatz ist die Kurve dagegen nach diesem Zeitpunkt fast horizontal.

Im kompletten Datensatz werden zwischen 8% und 12% der Fehler nicht behoben. Dabei ist der Anteil der nicht behobenen Fehlern bei den Contributoren nur leicht höher als bei den Entwicklern. Im reduzierten Datensatz werden wesentlich mehr Fehler nicht behoben. Auch sind die Unterschiede hier grösser. Die Hypothese C1 zur Wahrscheinlichkeit der Fehlerbehebung kann für den einfachen Datensatz mit Ausnahme der Differenz zwischen den offiziellen und den Kernentwicklern bestätigt werden. Für den vollständigen Datensatz stimmt sie zwar in der Tendenz, doch sind dort die Ergebnisse weniger eindeutig.

Grundsätzlich zeigen die Ergebnisse aber sehr hohe Kooperationsraten. Aus der umgekehrten Perspektive kann festgehalten werden, dass ungefähr 90% (im reduzierten Datensatz ungefähr 80%) der Fehler behoben werden. Dies ist umso erstaunlicher, wenn man bedenkt, dass die Fehlerbehebenden wohl in den vielen Fällen keinen direkten Nutzen aus der Fehlerbehebung ziehen. Die Fehler wurden ihnen ja von anderen Nutzern gemeldet, so dass die Vermutung nahe liegt, dass sie selbst nicht vom Fehler betroffen waren.

Die Ergebnisse dieser Schätzung sind mit Vorsicht zu geniessen, da keinerlei Kontrollvariablen berücksichtigt wurden. Diese Ergebnisse liefern aber erste Hinweise für eine Bestätigung der Hypothese C1, dass die Reputation und der Status des Fehlerberichterstatters die Zeit zur Fehlerbehebung verkürzt und die Chance erhöht, dass der Fehler behoben wird.

5.3.2 Cox-Regression

Mit Hilfe der Cox-Regression sollen die im vorangehenden Abschnitt gewonnen Erkenntnisse vertieft werden. Insgesamt wurden 10 verschiedene Modelle berechnet. Die Modelle unterscheiden sich in drei Komponenten. Erstens wurde je ein Modell mit dem kompletten und dem reduzierten Datensatz berechnet (Modelle M1, M3, M5, M7, M8, M10 vs. M2, M4, M6, M9). Zweitens wurde je ein Modell mit und ohne die Variablen zum Fehlerbehebenden berechnet (Modelle M3, M4, M8-10 vs. M1, M2, M5-7). Drittens wurde ein Modell mit allen Indikatoren berechnet und ein Modell nur mit denjenigen Indikatoren, welche im vollständigen Modell signifikante Effekte zeigten (Modelle M1-4 vs. M5-10). Zusätzlich wurde jedoch in zwei der so vereinfachten Modell die Art der Fehlerbehebung als zusätzliche Kovariate aufgenommen (Modelle M8, M10). Für die restlichen vereinfachten Modelle fehlt diese Information bzw. wurde sie im Subsample ausgeschlossen. Sie konnte deshalb nicht aufgenommen werden.

Zusätzlich zu diesen 8 Modellen, wurden noch zwei weitere Modelle zur Prüfung der Hypothese E2 geschätzt (Modelle M7 und M10). Diese Hypothese besagt, dass Fehler, welche nicht zu einem spezifischen Softwarepaket, sondern zu einem der generellen Pseudopakete gemeldet werden, weniger schnell behoben werden. Diese zwei Modelle wurden mit den gleichen Variablen wie die vereinfachten Modellen gerechnet. Jedoch wurde die Anzahl der Paketbetreuer durch Dummy-Variablen zu den Pseudopaketen ersetzt. Tabelle 5.4 zeigt die Ergebnisse der vollständigen Modelle und Tabelle A.8 diejenigen der vereinfachten Modelle und zu den Pseudopaketen.

In die vereinfachten Modelle wurde nur noch die Entwicklerkategorie als Reputationsindikator aufgenommen. Die Kooperationsvariablen zeigen sehr geringe Effekte, welche in den meisten Fällen nicht signifikant sind. Sie wurden deshalb ausgeschlossen. Sowohl der Status, wie auch die Aktivität zeigen im reduzierten Datensatz signifikante Effekte. Werden jedoch beide Variablen in das Modell eingefügt, dann verteilen sich die Effekte so, dass beide nicht mehr in allen Fällen signifikant sind. Dies kommt daher, dass beide etwas sehr ähnliches messen und stark miteinander korreliert sind (Spearman's $\rho = 0.85$). Auch erhöhte sich die erklärte Varianz (Pseudo R^2) nicht, wenn beide Variablen im Modell waren. Allgemein ist die erklärte Varianz im Modell mit allen Variablen nicht höher als im vereinfachten Modell.

Der Status wurde schlussendlich aus zwei Gründen bevorzugt. Erstens ist der Status einer Person für die Interaktionspartner einfacher zu erkennen als die bisherige Projektaktivität. Deshalb eignet sich der Status aus einer theoretischen Perspektive besser als Reputationsindikator. Zweitens ist die Interpretation der Resultate einfacher, da über die Dummy-Variablen der einzelnen Status der Unterschied zwischen diesem Status und dem Referenzstatus Contributor geschätzt wird. Die Interpretation einer Schätzung zur Aktivität ist dagegen weniger offensichtlich, da diese Variable aus mehreren anderen aggregiert wurde und zudem noch logarithmisch transformiert ist. Weiter wurde die Codierung des Schweregrades des Fehlers auf zwei Kategorien reduziert. Im vollständigen Modell hat sich gezeigt, dass insbesondere ein Unterschied zwischen den Graden, welche als Releasekritisch betrachtet werden und den tieferen Graden besteht. Deshalb wird der Schweregrad nur noch durch eine Dummy-Variable «Releasekritisch» einbezogen.

5.3.3 Reputation des Fehlerberichterstatters

Der Einfluss der Reputation des Fehlerberichterstatters zeigt sich in allen Modellen ohne die Kovariaten zum Fehlerbehebenden. Ich werde im Folgenden in diesem Abschnitt nur auf diese Modelle eingehen. Auf den Einfluss des Fehlerbehebenden und die Gründe, warum die Effekte in diesen Modellen nur noch teilweise vorhanden sind, werde ich im nächsten Abschnitt genauer eingehen.

In den vereinfachten Modellen (M5-7) sind die Reputationseffekte für alle Entwicklerkategorien sowohl im kompletten wie auch im reduzierten Datensatz signifikant. Auch in den Modellen mit allen Variablen (M1-2) zeigen sich für die Entwicklerkategorien und die Aktivität die erwarteten Effekte. Allerdings sind dort nur einige Effekte im reduzierten Datensatz signifikant. Dies hängt damit zusammen, dass alle Reputationsvariablen miteinander korreliert sind. Dadurch verteilen sich die Effekte auf die verschiedenen Variablen und sind je einzeln nicht mehr signifikant. Da diese Variablen jedoch aus theoretischer Sicht das gleiche zugrundeliegende Konstrukt, die Reputation des Fehlerberichterstatters, messen sollen, ist diese Verteilung nicht problematisch. Deshalb wurde im vereinfachten Modell, wie weiter oben ausgeführt, auch nur der Entwicklerstatus berücksichtigt.

Wie zu erwarten, ist der Effekt für die einfachen Entwickler am geringsten, gefolgt von den offiziellen Entwicklern und den Kernentwicklern. Jedoch sind diese Unterschiede zwischen den einzelnen Entwicklerstatus auch im vereinfachten Modell nicht signifikant. Währenddem die Unterschiede zwischen den Status in den einfachen Medianberechnungen ohne Kontrollvariablen noch signifikant waren, lassen sich hier keine eindeutigen Unterschiede mehr feststellen. Gleich wie in den Medianberechnungen sind die Unterschiede in der Tendenz im reduzierten Datensatz grösser als im vollständigen Datensatz.

Die im vollständigen Modell (M1-2) einbezogenen Variablen zum Umfang der Zusammenarbeit mit anderen Entwicklern zeigen nicht die zu erwartenden Effekte. Die Zusammenarbeit mit anderen Personen im Rahmen von Fehlerberichten hat überhaupt keinen eindeutigen Einfluss. Zumindest im hier verwendeten sehr einfachen Netzwerkmodell lassen sich keine Netzwerkeffekte nachweisen. Dies deutet darauf hin, dass sich Reputationsinformationen im Debian Projekt nicht primär über persönliche Kontakte verbreiten, sondern einfache, allgemein verfügbare Indikatoren wie der Entwicklerstatus

Tabelle 5.4: Effekte auf die Fehlerbehebungsrate (alle Variablen)

Datensatz	HT		M1 alle	M2 reduziert	M3 alle	M4 reduziert
Variablen Fehlerberichterstatter						
Status (Referenz Contributor)						
einfacher Entwickler	C2	+	1.08	1.07	1.11***	1.09*
offizieller Entwickler	C2	+	1.10	1.31**	0.98	1.00
Kernentwickler	C2	+	1.11	1.39*	0.97	0.99
Aktivität	C2	+	1.06	1.21*	0.95**	1.00
Zusammenarbeit						
Fehlerberichte	C3	+	1.00	0.99	1.00	1.00
Softwarepakete	C3	+	0.99	0.92*	1.04***	1.01
Dauer der Projektbeteiligung		+	0.97***	0.93***	1.01***	1.00
Variablen Fehlerbeheber						
Status (Referenz Contributor)						
einfacher Entwickler		+			1.71***	2.58***
offizieller Entwickler		+			1.71***	2.69***
Kernentwickler		+			1.90***	2.82***
Aktivität		+			1.00	1.16***
Zusammenarbeit						
Fehlerberichte		+			0.96***	0.96***
Softwarepakete		+			0.99*	0.98*
Dauer der Projektbeteiligung		+			1.03***	0.98***
Medianzeit Fehlerbehebung		–			0.95***	0.96***
gemeinsame Variablen						
vergangene Interaktion	A1	+			1.12***	1.08***
Variablen Fehlerbericht						
Schweregrad (Referenz Wishlist)						
Minor		+	1.10*	1.06	0.98	0.90***
Normal		+	1.11***	1.02	1.04*	1.04
Important		+	1.27***	1.28***	1.18***	1.34***
Serious		+	2.36***	3.26***	1.86***	2.81***
Grave		+	2.33***	3.13***	2.12***	2.89***
Critical		+	2.31***	3.48***	2.08***	2.71***
Anzahl Paketbetreuer	E1	–	0.97***	0.90***	1.00*	1.01
Worte im Fehlerbericht		+	0.92***	0.92***	0.96***	0.97**
Modellparameter						
N			210'791	97'184	162'203	68'428
Events			169'589	70'057	162'203	68'428
missing			94'551	34'198	142'139	62'954
Pseudo R^2			0.07	0.13	0.08	0.11
Clusters			20'642	12'038	17'526	8'983

Anmerkungen: Cox-Regression mit abhängiger Variable Zeit bis zur Fehlerbehebung; Maximum-Likelihood-Schätzung der Effekte auf die Hazardrate; Robuste Standardfehler mit Clustering für Fehlerberichterstatter (aus Platzgründen nicht ausgewiesen); Deskriptive Statistiken zu den Variablen im Anhang A.1; Signifikanzniveaus: * p<=0.05, ** p<=0.01, *** p<=0.001

Tabelle 5.5: Effekte auf die Fehlerbehebungsrate (vereinfachte Modelle ohne Fehlerbehebende

Datensatz	HT		M5 vollständig e^β	M5 vollständig S.E.	M6 reduziert e^β	M6 reduziert S.E.	M7 vollständig e^β	M7 vollständig S.E.
Variablen Fehlerberichterstatter								
Status (Referenz Contributor)								
einfacher Entwickler	C2	+	1.12**	0.04	1.15*	0.07	1.12**	0.04
offizieller Entwickler	C2	+	1.15***	0.03	1.40***	0.06	1.18***	0.03
Kernentwickler	C2	+	1.21***	0.04	1.50***	0.07	1.26***	0.04
Dauer der Projektbeteiligung		+	0.98***	<0.01	0.94***	0.01	0.98***	0.01
Variablen Fehlerbericht								
Releasekritisch		+	2.14***	0.03	3.34***	0.05	2.08***	0.02
Anzahl Paketbetreuer	E1	–	0.97***	<0.01	0.89***	0.01		
Pseudopaket (Referenz normales Paket)								
Generell	E2	–					0.42***	0.15
Team	E2	–					0.94**	0.02
Worte im Fehlerbericht		+	0.93***	0.01	0.92***	0.02	0.91***	0.01
Modellparameter								
N			210'791		97'184		303'794	
Events			169'589		70'057		240'418	
missing			94'551		34'198		1'548	
Pseudo R^2			0.07		0.13		0.07	
Clusters			20'642		12'038		27'783	

Anmerkungen: Cox-Regression mit abhängiger Variable Zeit bis zur Fehlerbehebung; Maximum-Likelihood-Schätzung der Effekte auf die Hazardrate; S.E.: Robuste Standardfehler mit Clustering für Fehlerberichterstatter; Deskriptive Statistiken zu den Variablen im Anhang A.1; Signifikanzniveaus: * p<=0.05, ** p<=0.01, *** p<=0.001

Tabelle 5.6: Effekte auf die Fehlerbehebungsrate (vereinfachte Modelle mit Fehlerbehebenden)

Datensatz	HT		M8 vollständig e^β	M8 vollständig S.E.	M9 reduziert e^β	M9 reduziert S.E.	M10 vollständig e^β	M10 vollständig S.E.
Variablen Fehlerberichterstatter								
Status (Referenz Contributor)								
einfacher Entwickler	C2	+	1.08***	0.02	1.12***	0.03	1.06**	0.02
offizieller Entwickler	C2	+	0.99	0.02	1.04	0.03	0.97	0.02
Kernentwickler	C2	+	1.01	0.02	1.08*	0.03	1.00	0.02
Dauer der Projektbeteiligung		+	1.01**	<0.01	1.00	<0.01	1.01*	<0.01
Variablen Fehlerbeheber								
Status (Referenz Contributor)								
einfacher Entwickler		+	1.60***	0.02	2.71***	0.06	1.53***	0.02
offizieller Entwickler		+	1.55***	0.02	2.80***	0.06	1.54***	0.02
Kernentwickler		+	1.64***	0.02	3.03***	0.06	1.68***	0.02
Dauer der Projektbeteiligung		+	1.02***	<0.01	0.98***	<0.01	1.03***	<0.01
Medianzeit Fehlerbehebung		–	0.95***	<0.01	0.96***	<0.01	0.96***	<0.01
gemeinsame Variablen								
vergangene Interaktion	A1	+	1.07***	0.01	1.08***	0.01	1.07***	0.01
Variablen Fehlerbericht								
Releasekritisch		+	1.85***	0.04	2.70***	0.03	1.83***	0.04
Anzahl Paketbetreuer	E1	–	1.01***	<0.01	1.01	0.01		
Pseudopaket (Referenz normales Paket)								
Generell	E2	–					1.97**	0.23
Team	E2	–					1.44***	0.02
Worte im Fehlerbericht		+	0.97***	0.01	0.98*	0.01	0.97***	0.01
Art der Fehlerbehebung (Referenz Softwarepaket)								
Mail			0.80***	0.01			0.81***	0.01
«Wontfix»			0.81***	0.03			0.83***	0.02
Entfernung			0.48***	0.11			0.76***	0.04
Modellparameter								
N			168'460		69'442		239'006	
Events			168'460		69'442		239'006	
missing			136'882		61'940		66'336	
Pseudo R^2			0.09		0.11		0.09	
Clusters			17'923		9'068		23'595	

Anmerkungen: Cox-Regression mit abhängiger Variable Zeit bis zur Fehlerbehebung; Maximum-Likelihood-Schätzung der Effekte auf die Hazardrate; Robuste Standardfehler mit Clustering für Fehlerberichterstatter in Klammern; Koeffizienten jeweils e^β; Deskriptive Statistiken zu den Variablen im Anhang A.1; Signifikanzniveaus: * p<=0.05, ** p<=0.01, *** p<=0.001

eine grössere Bedeutung haben.

Auch wenn gewisse Effekte zur Zusammenarbeit signifikant sind, wenn auch in die entgegengesetzt Richtung, sind Überlegungen zu den Gründen für diese Effekte primär Spekulation. Die Bedeutung dieser Effekte sollte nicht überbewertet werden, da sie je nach genauer Ausgestaltung des Modells und der einbezogenen Variablen sehr instabil sind und auch das Vorzeichen wechseln.

Die Dauer der Projektbeteiligung wurde als Kontrollvariable für die Erfahrung des Fehlerberichterstatters eingefügt. Entgegen den Erwartungen hat die Erfahrung einen negativen Effekt auf die Fehlerbehebung. Pro zusätzliches Jahr der Projektbeteiligung beträgt der Effekt jedoch nur 2% bis 6%. Der Effekt verschwindet in den Modellen ohne die zensierten Fälle. Deshalb sollte dieser Effekt, obwohl er klar signifikant ist, nicht überbewertet werden. Die Dauer der Projektbeteiligung scheint jedoch kein guter Proxy für die Qualität der entsprechenden Fehlerberichte und die Erfahrung des Fehlerberichterstatters zu sein. Zu beachten ist auch, dass die Dauer der Projektbeteiligung mit dem Entwicklerstatus korreliert ist (Spearman's $\rho = 0.52$).

5.3.4 Einfluss des Fehlerbehebers

Die Vermutung liegt nahe, dass Eigenschaften der Person, welche einen Fehler behebt einen grösseren Einfluss auf die Zeit zur Fehlerbehebung haben als Eigenschaften des Fehlerberichterstatters. Die Erfahrung, das technische Können, das Interesse an der Fehlerbehebung und die verfügbare Zeit des Fehlerbehebers werden dabei von Bedeutung sein. Nicht alle, aber einige dieser Einflüsse können mit den vorhandenen Daten modelliert werden. Zu den Fehlerbehebenden wurden jeweils die gleichen Variablen wie zu den Fehlerberichterstattern ins Modell aufgenommen. Dies macht die Ergebnisse besser vergleichbar.

Die Variablen werden jedoch für die Fehlerbehebenden anders interpretiert. Reputation als Koordinationsmechanismus spielt für sie keine Rolle, da die Kooperation des Fehlerberichterstatters durch das Melden des Fehlers bereits erfolgt ist. Sie sind allenfalls am Aufbau der eigenen Reputation für zukünftige Interaktionen interessiert. Die Variablen messen für sie vielmehr die Erfahrung und die technische Kompetenz. Es ist zu erwarten, dass Personen, welche einen höheren Entwicklerstatus haben, eine grösseren Aktivität aufweisen oder länger am Projekt beteiligt sind über eine grössere Erfahrung und Kompetenz im Beheben von Fehlern verfügen. Erfahrung und technisches Können sind stark miteinander verknüpft. Die beiden Eigenschaften können in den Variablen nicht scharf voneinander getrennt werden. Die Dauer der Projektmitgliedschaft misst jedoch eher die Erfahrung und bei der Entwicklerkategorie liegt der Schwerpunkt eher auf der technischen Kompetenz. Der Aktivitätsindex und die Kooperationsvariablen sind hingegen nicht klar zuzuordnen. Zusätzlich wurde für die Fehlerbeheber noch die Medianzeit der bisher behobenen Fehler ins Modell eingefügt.

Die Modellschätzungen (Modelle M3-4 und M8-10) zeigen, dass der Entwicklerstatus einen starken positiven Einfluss auf die Fehlerbehebung hat und die Medianzeit einen negativen. Die restlichen Variablen zeigen einen wesentlich schwächeren oder weniger eindeutigen Einfluss. Der Entwicklerstatus hat einen stärkeren Einfluss als die Dauer

der Projektbeteiligung. Dies ist ein Hinweis darauf, dass die technische Kompetenz eine grössere Rolle spielt, als die Erfahrung. Die Unterschiede zwischen den einzelnen Entwicklerstatus sind aber nicht signifikant. Die Effekte zur Entwicklerkategorie bedeuten deshalb lediglich, dass ein Unterschied zwischen den Contributoren und den restlichen Kategorien besteht. Das Ergebnis, dass technische Kompetenz eine grössere Rolle spielt, muss aus mehreren Gründen mit Vorsicht betrachtet werden. Erstens sollte der theoretische Zusammenhang der untersuchten Variablen zu den Konzepten technischen Kompetenz und Erfahrung noch genauer untersucht werden. Zweitens ist die Zahl der Fehler gering, welche durch Contributoren behoben werden. Drittens kann der Unterschied auch in der Art der Fehler liegen, die von Contributoren behoben werden. So können Contributoren keine Fehler beheben, die eine Verbesserung eines Softwarepakets bedingen. Zur Kontrolle dieses Effekts enthalten die Modelle M8 und M10 die Art der Fehlerbehebung als Proxy für die Art des Fehlers. Die Effekte auf die Entwicklerkategorie werden dadurch im Vergleich zum Modell M3 nur leicht verringert. Die Art der Fehlerbehebung erklärt also nur einen kleinen Teil des Unterschiedes. Ein vierter Grund zur Vorsicht ist, dass die Kategorie des Fehlerbehebers zum Zeitpunkt bestimmt wird, zu dem der Fehler gemeldet wird und nicht zum Zeitpunkt, zu dem er den Fehler tatsächlich behebt.

Die vorangehenden Überlegungen haben gezeigt, dass die Modelle mit den Variablen zu den Fehlerbehebern aus verschiedenen Gründen mit Vorsicht zu betrachten sind. Dazu gibt es insbesondere drei Gründe: Erstens fehlen im Datensatz relevante Informationen zu den Fehlerbehebern. Ihre technische Kompetenz, die Zeit, die ihnen für die Arbeit am Projekt momentan zur Verfügung steht und ihr Interesse an der Behebung des Fehlers sind nicht oder nur über Proxys bekannt. Falls jedoch nur gewisse der relevanten Einflussgrössen und diese nur mit einer geringen Genauigkeit ins Modell aufgenommen werden, dann steigt die Wahrscheinlichkeit von Verzerrungen.

Zweitens wurden die Variablen zum Fehlerbeheber zum Zeitpunkt erhoben, zu dem der Fehler gemeldet wurde. Dies kann insbesondere bei Fehlern, welche über längere Zeit nicht behoben werden, zu Verzerrungen führen. So kann ein Fehlerbeheber inzwischen die Kategorie gewechselt haben. Dies erklärt auch, warum teilweise auch Fehler durch ein verbessertes Paket behoben wurden, welche von einem Contributor bearbeitet wurden. Allerdings wäre eine Erhebung zum Zeitpunkt der Fehlerbehebung auch nicht unproblematisch, da dann die Variablen zum Fehlerberichterstatter und zum Fehlerbeheber nicht mehr zum gleichen Zeitpunkt erhoben werden. Zudem ist die Fehlerbehebung ein Prozess. Es kann deshalb nicht ein genauer Zeitpunkt festgemacht werden, zu dem die Variablen einen Einfluss haben. Ohne genauere theoretische Fundierung bleibt jedenfalls unklar, welcher Zeitpunkt zur Bestimmung der Variablen am geeignetsten ist.

Drittens fehlen in allen Modellen mit den Fehlerbehebenden die zensierten Fälle. Für zensierte Fälle, also Fehler, welche zum Zeitpunkt des Untersuchungsendes noch nicht behoben waren, ist auch nicht bekannt, wer sie behoben hat. Es können deshalb keine Werte für diese Variablen bestimmt werden. Alle Effekte, welche primär dadurch entstehen, dass gewisse Fehler überhaupt nicht behoben werden, sind in diesen Modellen nicht repräsentiert. Das Verschwinden des Einflusses des Status des Fehlerberichterstatters kann darauf zurückzuführen sein. Das würde bedeuten, dass Fehlern von Personen mit tiefer Reputation mit einer geringeren Wahrscheinlichkeit überhaupt behoben werden.

Es kann nicht davon ausgegangen werden, dass das Entfernen der zensierten Fälle zu keinen systematischen Verzerrungen führt. Auch ist unklar, ob die Schätzung der Einflüsse mit einer Cox-Regression noch angemessen ist. Die Modelle mit den Variablen zu den Fehlerbehebenden müssen deshalb als grundlegend Unterschiedlich zu den Modellen ohne diese Variablen angesehen werden. Sie können nur sehr beschränkt verglichen werden.

Das vereinfachte Modell mit den Fehlerbehebenden wurde zusätzlich mit einer OLS-Regression geschätzt. Die Effekte bleiben sich im Grossen und Ganzen gleich. Im Unterschied zur Cox-Regression ergibt sich jedoch ein signifikanter Effekt, falls der Fehlerberichterstatter ein Entwickler ist. Der geschätzte Unterschied beträgt jedoch nur etwas mehr als 7 Tage.

5.3.5 Wiederholte Interaktion

Der positive Effekt wiederholter Interaktion lässt sich in allen Modellen bestätigen. Falls der Fehlerberichterstatter und der Fehlerbehebende bereits einmal miteinander interagiert haben, dann hat dies je nach Modell einen Einfluss von 6% bis 12% auf die Hazardrate. Wird mit einer Kaplan-Meier Schätzung lediglich der Einfluss der vergangenen Interaktion untersucht, so reduziert sich die Medianzeit zur Fehlerbehebung im vollständigen Datensatz von 37.7 auf 18.5 Tage, im reduzierten Datensatz von 18.8 auf 10.9 Tage. Im vollständigen Datensatz wird damit die Medianzeit mehr als halbiert. Die im Vergleich zu den Schätzungen der Medianzeiten in Tabelle 5.3 wesentlich tieferen Zeiten sind dadurch bedingt, dass in den Schätzungen zur vergangenen Interaktion die zensierten Fälle fehlen.

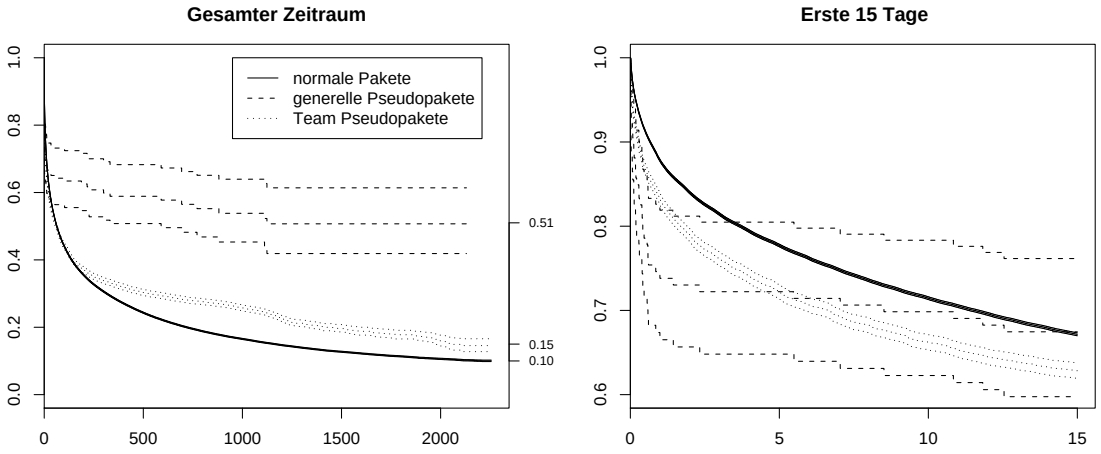
Der positive Effekt der vergangenen Interaktion ist insbesondere deshalb von Bedeutung, da die vergangene Interaktion eine der wenigen Variablen im Datensatz ist, welche direkt beobachtbar ist. Sie ist im Unterschied zu den Variablen zum Fehlerbeheber und in einem geringeren Umfang auch den Variablen zur Reputation des Fehlerberichterstatters wesentlich weniger mit Unsicherheiten darüber behaftet, ob mit den entsprechenden Variablen auch wirklich das gemessen werden kann, das gemessen werden soll.

5.3.6 Verantwortungsdiffusion

Ausgehende vom Freiwilligendilemma habe ich die Hypothesen E1 & E2 zur Verantwortungsdiffusion formuliert. In fast allen Modellen zeigen sich signifikante Effekte für die Verantwortungsdiffusion. Einzig in den Modellen mit den Fehlerbehebern (M3-4 und M8-10) zeigen sich keine beziehungsweise für die Pseudopakete sogar umgekehrte Effekte. Die Überlebenskurven in Abbildung 5.8 zeigen jedoch klar, dass die Zensierung nicht unabhängig davon ist, ob der Fehler zu einem normalen Paket oder zu einem Pseudopaket gemeldet wurde. Diese Modelle, welche die zensierten Fälle nicht enthalten, sind deshalb nicht verwendbar.

Hypothese E1 postuliert, dass die Anzahl der für einen Fehlerbericht potentiell zuständigen Personen einen negativen Einfluss auf die Wahrscheinlichkeit der Fehlerbehebung hat. Im Datensatz zeigt sich dieser Einfluss klar. Ein zusätzlicher Entwickler, der für das betroffene Softwarepaket zuständig ist, verringert die Hazardrate im vollständigen Datensatz um 3%, im reduzierten Datensatz sogar um 11%. Wie aus dem Freiwilligendilemma

Abbildung 5.8: Überlebenskurven nach Pseudopaketen



Anmerkung: N=305'342, normale Pakete 294'618 (233'800 Ereignisse), generelle Pseudopakete 126 (57), Team Pseudopakete 10'598 (7804), Überlebenskurve mit 95% Konfidenzintervall

vorhergesagt, verringert also die Anzahl der potentiellen Fehlerbeheber die Chance, dass ein Fehlerbericht behoben wird.

Die zweite Hypothese zum Freiwilligendilemma (E2) postuliert den gleichen Effekt für Fehler gegen Pseudopakete. Diese Pakete haben nicht wie die normalen Softwarepakete einen oder mehrere zuständige Paketbetreuer, sondern werden an Mailinglisten weitergeleitet. Eine Person, welche die Mailingliste liest, sollte sich dann darum kümmern. Damit ist der Kreis der potentiell zuständigen Personen noch weit diffuser als bei Paketen mit mehreren Paketbetreuern. Die Empfänger des Fehlerberichts wissen nicht einmal genau, wer sonst den Fehlerbericht noch bekommen hat. Wie aus diesem Grund zu erwarten war, zeigt sich insbesondere beim Pseudopaket «general» ein noch deutlicherer Effekt. Für Fehler zum Pseudopaket «general» reduziert sich die Hazardrate um 58%. Für Fehler gegen Pseudopakete für welche ein Team zuständig ist, reduziert sie sich um 6%.

Eine genauere Betrachtung der Überlebenskurven (Abbildung 5.8) zeigt jedoch ein etwas differenzierteres Bild. Wie aus der Cox-Regression zu erwarten ist, unterscheiden sich die Kurven stark. Es ist deutlich zu sehen, dass die Fehler zu den Pseudopaketen mit einer geringeren Wahrscheinlichkeit behoben werden. Werden jedoch nur die ersten 15 Tage der Überlebenskurve betrachtet, dann zeigt sich dort ein umgekehrtes Bild. Die Fehler der Pseudopakete werden in dieser Zeitspanne schneller behoben. Insbesondere die Fehler zu dem Pseudopakete «general». Dies erklärt auch die Umkehrung der Effekte im Cox-Modell ohne die zensierten Fälle.

Es sind zwei sich nicht ausschliessende Erklärungen für diese Tatsache denkbar. Erstens werden zu Pseudopaketen oft sehr unspezifische Fehler gemeldet. Bei einigen dieser Fehler handelt es sich aber nicht um Fehler in der Software, sondern um Benutzerfehler. Diese können ohne dass ein Entwicklungsaufwand notwendig wäre, einfach wieder geschlossen

werden. Die «Fehlerbehebung» ist also wesentlich einfacher. Zweitens führt die Tatsache, dass sehr viele Personen potentiell für den Fehler zuständig sind, nicht nur zu Verantwortungsdiffusion, sondern erhöht auch die Wahrscheinlichkeit, dass jemand den Fehler sieht und sich sofort um diesen kümmert. Falls dies passiert, werden diese Fehler sehr schnell behoben. Falls sich jedoch nicht sofort jemand darum kümmert, dann sinkt aufgrund der Verantwortungsdiffusion die Chance, dass sich zu einem späteren Zeitpunkt doch noch jemand dem Fehlerbericht annehmen wird.

5.3.7 Eigenschaften des Fehlerberichts

Zum Abschluss der Analyse der Cox-Modelle sollen noch einige Bemerkungen zu den fehlerberichtsbezogenen Kontrollvariablen angefügt werden. Als Kontrollvariablen wurden der Schweregrad des Fehlerberichts und die Länge des Fehlerberichts in Worten verwendet.

In den bisherigen Betrachtungen standen, wie dies für eine soziologische Untersuchung zu erwarteten ist, die sozialen Aspekte der Fehlerbehebung im Vordergrund. Der Schwerpunkt lag auf der Untersuchung der Kooperation zwischen dem Fehlerberichterstatter und dem Fehlerbehebenden. Aus der Innensicht des Open Source Projektes stehen jedoch primär die technischen Aspekte im Vordergrund. Mit dem Schweregrad des Fehlers können diese ins Modell integriert werden. Der Schweregrad ist ein Hinweis darauf, wie stark der Fehler die Benutzung des betroffenen Softwarepakets einschränkt. Fehler mit Grad «serious» oder höher verhindern zudem, dass das Paket in den «testing» Zweig kommt, aus dem das nächste Release der Distribution bestehen wird. Diese Fehler werden deshalb als releasekritisch bezeichnet. Es ist zu erwarten, dass Fehler mit einem höheren Schweregrad eine höhere Priorität geniessen. Der Schweregrad des Fehlerberichts ist die wichtigste im Datensatz verfügbare Kontrollvariable, um das Ergebnis verfälschende Einflüsse der technischen Seite des Fehlerberichts zu kontrollieren.

In die vollständigen Modelle (M1-4) wurde jeder der 7 möglichen Schweregrade als Dummy-Variable integriert. Wie zu erwarten, hat ein höherer Schweregrad einen positiven Effekt auf die Hazardrate. Der Effekt erhöht sich jedoch nicht linear mit dem Schweregrad, sondern hat einen grösseren Sprung zwischen den nicht-releasekritischen und den releasekritischen Fehlern. In den vereinfachten Modellen (M5-10) wurde nur noch diese Unterscheidung beachtet. Dort wird die Hazardrate für releasekritische Fehler im vollständigen Datensatz (M5) mehr als verdoppelt, im reduzierten Datensatz (M6) sogar mehr als verdreifacht. Der Einfluss des Schweregrades ist damit wesentlich grösser als der Einfluss jeder anderen Variablen mit Ausnahme des Pseudopakets «general».[1] Dies bedeutet, dass technische Aspekte einen weit grösseren Einfluss auf die Fehlerbehebung haben als die sozialen.

Die Länge des Fehlerberichts wird als Proxy für die Ausführlichkeit der Fehleranalyse in das Modell eingefügt. Eine ausführlichere Fehleranalyse vereinfacht die Fehlerbehebung, da der Fehlerbericht bereits alle relevanten Informationen enthält und nicht zuerst weitere Details erfragt werden müssen. Erstaunlicherweise hat jedoch die logarithmierte Länge

[1] Die Einflüsse der nicht Dummy-Variablen können nicht direkt verglichen werden, da diese Variablen nicht die gleiche Einheit haben. Sie können aber, auch wenn sie ihren Maximalwert erreichen, keinen grösseren Einfluss haben.

des Fehlerberichts in Worten einen negativen Einfluss auf die Hazardrate. Ein längerer Fehlerbericht verlängert damit die Zeit bis zur Fehlerbehebung und reduziert deren Chance. Die wahrscheinlichste Erklärung für diesen Effekt ist, dass die Länge des Fehlerberichts kein guter Proxy für die Güte der Fehleranalyse ist. Vermutlich ist sie eher ein Proxy für die Komplexität des Fehlers. Um einen komplexeren Fehler zu beschreiben, braucht es mehr Platz. Da jedoch die Qualität des Fehlerberichts eine der wesentlichen Einflussgrössen ist und diese mit dem vorliegenden Datensatz nur schwer messbar ist, sollten für weitere Untersuchungen bessere Indikatoren dazu gefunden werden.

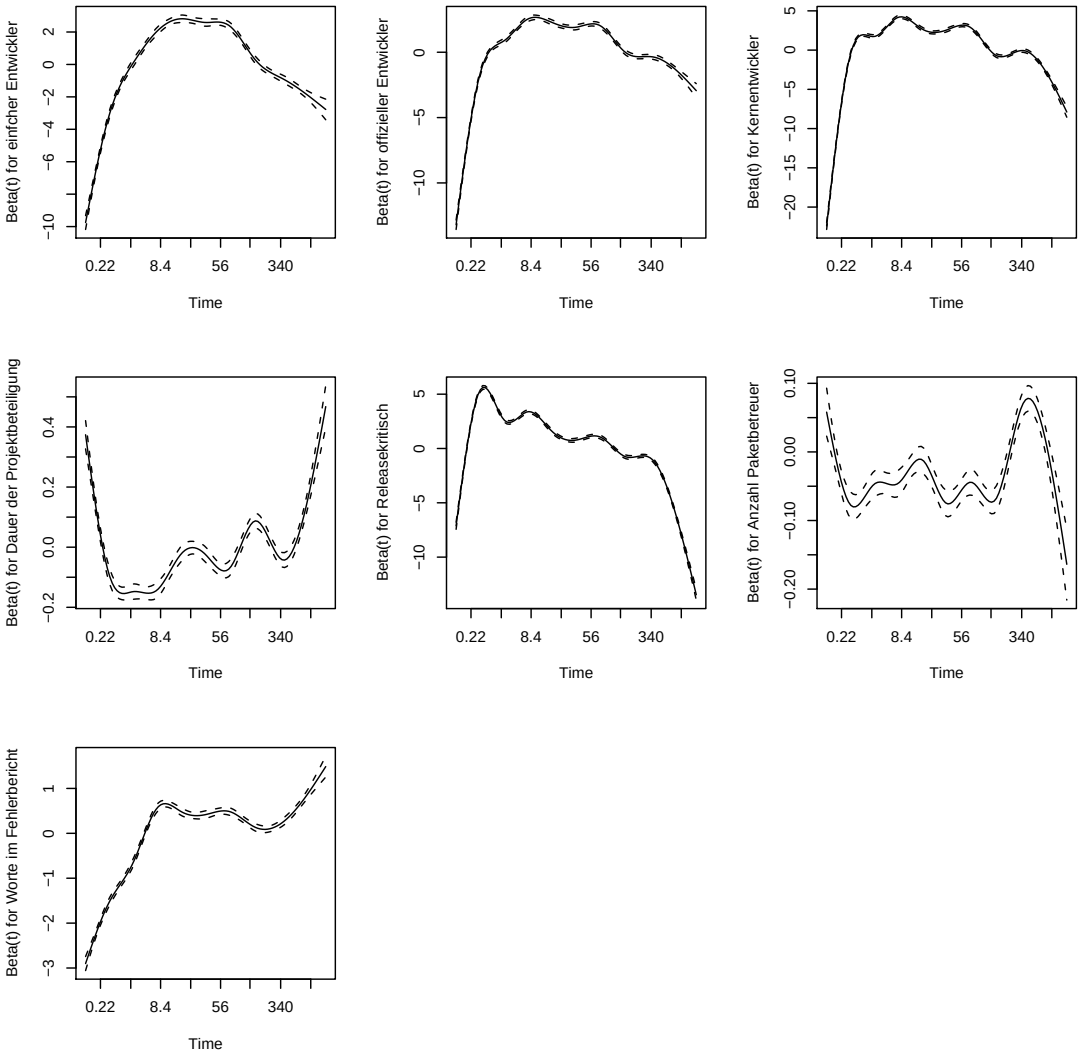
5.3.8 Proportionalitätsannahme der Cox-Regression

Ich werde die Proportionalität der Kovariateneinflüsse lediglich für das reduzierte Modell ohne die Fehlerberichterstatter im vollständigen Datensatz (M5) exemplarisch untersuchen. Die Ergebnisse für die anderen Modelle unterscheiden sich nicht wesentlich.

Abbildung 5.9 zeigt die geglätteten Kurven durch die Schoenfeld Residuen für jede Kovariate. Diese zeigen eindeutig, dass die Proportionalität für keine der Variablen erfüllt ist. Eine genauere Betrachtung der Kurven zeigt, dass sich einige der zeitabhängigen Einflüsse durch einfache Überlegungen erklären lassen. Die drei Dummy-Variablen für den Entwicklerstatus in der obersten Zeile der Abbildung zeigen, dass der Effekt des Entwicklerstatus ganz zu Beginn klein ist und sich insbesondere in der Mitte zeigt. Mit zunehmender Zeit nimmt er wieder ab. Offensichtlich spielt Reputation bei den Fehlern, die sehr schnell behoben werden, keine Rolle und die Entwicklerkategorie hat sogar einen negativen Effekt. Dies ist im Kontrast zur Referenzkategorie des Contributors zu sehen. Wahrscheinlich berichten diese vermehrt Fehler, die eigentlich gar keine Fehler, sondern Benutzerfehler sind und die sofort wieder geschlossen werden können. Dass der Effekt der Reputation mit der Zeit abnimmt ist auch aus der Theorie erklärbar. Falls das Reputationssignal nach einer gewissen Zeit nicht zur Behebung des Fehlers motiviert hat, dann ist es naheliegend, dass dies auch später nicht der Fall sein wird. Reputation verliert ihre Wirkung, wenn ein Fehler längere Zeit nicht behoben wurde. Ähnlich verhält es sich mit den releasekritischen Fehlern. Auch diese Eigenschaft eines Fehlers verliert mit der Zeit ihre beschleunigende Wirkung. Lange nicht behobene releasekritische Fehler sind oft Fehler, die aus projektexternen Gründen nicht einfach behoben werden können. Beispielsweise Probleme mit der Lizenz eines Softwarepakets.

Im Anhang A.2 befindet sich eine Schätzung der vereinfachten Modelle (M5-6 und M8-9) mit einer parametrischen Regression. Diese zeigt in der Richtung der Effekte die gleichen Resultate wie die Cox-Regression. Ich gehe deshalb davon aus, dass die Verletzung der Proportionalitätsannahme nicht zu einer Umkehr der Effekte führt. Die im Cox-Modell geschätzten Parameter sind dabei wohl als Durchschnittswerte über die Zeit zu verstehen. Die in den vorangehenden Abschnitten dargestellten Ergebnisse behalten deshalb ihre Gültigkeit. Für eine genauere Analyse und um das Ausmass und die Wirkung der Reputationseffekte besser verstehen zu können, müssten jedoch weitere Analysen angestellt werden, welche den Rahmen dieser Arbeit sprengen würden. Dabei könnte einerseits der Versuch unternommen werden, ein der Fehlerbehebung angemessenes parametrisches Modell zu finden oder durch geeignete Kombination der Kovariaten mit

Abbildung 5.9: Graphische Darstellung der Schoenfeld Residuen für Modell M5



Anmerkung: Plot geglätteter Kurven durch die Schoenfeld-Residuen für Modell M5 in Tabelle 5.5. Die Kurven wurden mit 10 Freiheitsgraden berechnet. Eine grössere Zahl der Freiheitsgrade bedeutet eine «feinere» Approximation. Gestrichelte Linien im Abstand von 2 * Standardfehler; Ein Proportionalitätstest ergibt für alle Variablen signifikante Abweichungen von der Proportionalitätsannahme; Berechnet mit der R Funktion cox.zph

der Prozesszeit die Zeitabhängigkeit in das Cox-Modell integriert werden.

5.3.9 Beachtung der Fehlerberichte

Unter den Hypothesen zu den Reputationsprämien wurde nicht nur behauptet, dass die Reputation die Wahrscheinlichkeit der Fehlerbehebung erhöht (Hypothese C1) und die Zeit bis zur Fehlerbehebung verkürzt (Hypothese C2), sondern auch, dass sich mehr Entwickler um einen Fehler kümmern. Dieser Zusammenhang soll aus den in Abschnitt 4.4.2 dargelegten Gründen mit einer Poisson Regression mit der Anzahl der an der Fehlerbehebung beteiligten Personen als abhängiger Variable untersucht werden. Die Reputation wurde mit den gleichen Variablen modelliert, welche auch im vereinfachten Cox-Modell verwendet wurden.

Tabelle 5.7: Effekte auf die Beachtung der Fehlerberichte

Datensatz			**M11 vollständig**	
	HT		e^{β}	S.E.
Status (Referenz Contributor)				
einfacher Entwickler	C4	+	0.97**	0.01
offizieller Entwickler	C4	+	0.96***	0.01
Kernentwickler	C4	+	0.95***	0.01
Dauer der Projektbeteiligung			1.00	0.001
Releasekritisch		+	1.26***	0.01
Überlebenszeit		+	1.11***	0.002
Anzahl Paketbetreuer		+	1.04***	0.001
Fehler behoben		+	1.46***	0.01
Konstante			1.54***	0.01
N			210’791	
missing			94’551	
Clusters			20’642	

Anmerkungen: Poisson Regression (Maximum-Likelihood-Schätzung) mit abhängiger Variable Anzahl beteiligte Personen; robuste Standardfehler mit Clustering für den Fehlerberichterstatter (Bootstrapping); Überlebenszeit in Jahren; Deskriptive Statistiken zu den Variablen im Anhang A.1; Signifikanzniveaus: * p<=0.05, ** p<=0.01, *** p<=0.001

Die in Tabelle 5.7 dargestellten Ergebnisse stützen diese Hypothese jedoch nicht. Die Hypothese kann nur am vollständigen Datensatz geprüft werden, da im reduzierten Datensatz die Fehler ausgeschlossen wurden, an denen mehr als zwei Personen (Berichterstatter und fehlerbehebender Entwickler) beteiligt sind. Der Status des Fehlerberichterstatters hat sogar den umgekehrten Einfluss. Er verringert die Anzahl der Personen, welche sich an der Fehlerbehebung beteiligen. Der Einfluss ist jedoch relativ gering. Auch die Zeit, die ein Fehler offen war, spielt nur eine untergeordnete Rolle. Damit der Effekt überhaupt

Tabelle 5.8: Fehlerbehebung nach Entwicklerstatus

Datensatz	**vollständig**		**reduziert**	
Contributor	13'753	6%	1'342	1%
einfache Entwickler	27'478	11%	9'546	11%
offizielle Entwickler	60'078	25%	25'549	28%
Kernentwickler	138'930	57%	52'748	59%
missing	1'422	1%	732	1%

Anmerkung: 63'681 bzw. 41'465 Fehlerberichte waren zum Zensierungszeitpunkt noch nicht behoben. Diese werden in der Tabelle nicht aufgeführt.

sichtbar wird, muss die Überlebenszeit in Jahre umgerechnet werden. Ein zusätzliches Jahr, in dem der Fehler offen war, erhöht die Chance, dass sich eine weitere Person beteiligt um 11%. Einen wesentlichen Einfluss hat lediglich der Schweregrad des Fehlers und die Tatsache, dass der Fehler bereits behoben wurde. Beim Schweregrad zeigt der Effekt in die erwartete Richtung. Releasekritische Fehler haben eine um 26% erhöhte Wahrscheinlichkeit, dass sich eine zusätzliche Person an der Behebung beteiligt. Dass sich um bereits behobene Fehler mehr Personen gekümmert haben, ist rein technisch zu begründen. Um einen behobenen Fehler müssen sich mindestens zwei Personen gekümmert haben. Der Fehlerberichterstatter und der Fehlerbehebende. Offene Fehler, welche keine grössere Beachtung finden, bestehen jedoch oft nur aus dem Fehlerbericht, ohne dass irgendeine andere Person den Fehler beachtet hätte.

Aufgrund der Ergebnisse müssen auch die theoretischen Zusammenhänge, welche zu dieser Hypothese geführt haben, nochmals überdacht werden. Reputation wird als Signal für die Kooperationsbereitschaft eines Akteurs verstanden. Sie soll andere Akteure zu (indirekt) reziproker Kooperation motivieren. Eine hohe Reputation müsste deshalb mehr Entwickler zur Kooperation und damit zur Hilfe bei der Fehlerbehebung anregen. Soweit die theoretische Überlegung. Dieser einfache Mechanismus macht jedoch nur Sinn, wenn die Hilfe zusätzlicher Personen auch wirklich zu einer schnelleren Fehlerbehebung beiträgt. Eine im Kontext der Softwareentwicklung oft zitierte Weisheit besagt jedoch, dass die Entwicklungsgeschwindigkeit nur sehr begrenzt durch weitere Entwickler erhöht werden kann.[2] Falls demnach ein Entwickler sieht, dass sich bereits jemand um den Fehler kümmert, so wird er sich hüten, sich auch noch einzumischen und damit die Fehlerbehebung eher zu verlangsamen. So könnte dieses Ergebnis auch darin begründet sein, dass die zugrundeliegende Hypothese falsch ist.

5.3.10 Status des Fehlerbehebenden

Neben dem Reputationseinfluss auf die eigentliche Fehlerbehebung habe ich auch einen Einfluss der Reputation des Fehlerberichterstatters auf den Status des Fehlerbehebenden

[2]Diese Tatsache ist in der Informatik als Brooks-Law oder «Mythical Man-Month» bekannt (Brooks 1975).

Tabelle 5.9: Effekte auf den Status des Fehlerbehebenden

Datensatz			**M12 vollständig**		**M13 reduziert**	
	HT		e^{β}	S.E.	e^{β}	S.E.
Status (Referenz Contributor)						
einfacher Entwickler	D1	+	1.07	0.04	0.99	0.06
offizieller Entwickler	D1	+	1.21***	0.03	1.32***	0.05
Kernentwickler	D1	+	1.19***	0.04	1.24***	0.06
Dauer der Projektbeteiligung			0.98***	0.01	0.95***	0.06
Releasekritisch		+	1.37***	0.03	1.27***	0.01
Art der Fehlerbehebung (Referenz Softwarepaket)						
Mail			0.53***	0.02		
«Wontfix»			0.47***	0.05		
Entfernung			5.47***	0.21		
Konstante			6.03***	0.02	7.64***	0.03
N			239'006		88'769	
missing			66'336		42'613	
Clusters			23'595		10'800	

Anmerkungen: Logit-Regression mit abhängiger Variable Fehlerbeheber ist ein offizieller Entwickler oder Kernentwickler; robuste Standardfehler mit Clustering für den Fehlerberichterstatter (Bootstrapping); Deskriptive Statistiken zu den Variablen im Anhang A.1; Signifikanzniveaus: * p<=0.05, ** p<=0.01, *** p<=0.001;

postuliert (Hypothese D). Diese Effekte sollen als nächstes untersucht werden. Als abhängige Variable wurde eine Dummy-Variable verwendet. Diese hat die folgenden Ausprägungen: Der Fehlerbeheber ist entweder ein offizieller Entwickler oder ein Kernentwickler und der Fehlerbeheber ist ein Contributor oder ein einfacher Entwickler. Die Trennlinie liegt damit zwischen den einfachen Entwicklern und den offiziellen Entwicklern. Dies wurde insbesondere so gewählt, weil nur sehr wenige Fehlerberichte von Contributoren behoben werden. Auch ergäben sich sonst systematische Verzerrungen, da Contributoren nur solche Fehler beheben können, für deren Korrektur kein verbessertes Softwarepaket benötigt wird. Tabelle 5.8 gibt einen Überblick über die Zahl der Fehler, welche durch die jeweilige Entwicklerkategorie behoben werden. Auch mit dieser Trennlinie wird der überwiegende Teil der Fehler von einem offiziellen Entwickler oder einem Kernentwickler behoben.

Tabelle 5.9 zeigt die Ergebnisse der Logit-Regression. Die erwarteten Effekte sind nur für die offiziellen Entwickler und die Kernentwickler signifikant. Im vollständigen Datensatz wird die Signifikanz für die einfachen Entwickler nur knapp verfehlt ($p = 0.054$). Im reduzierten Datensatz zeigt sich hingegen für die einfachen Entwickler kein Effekt. Ein offizieller Entwickler oder ein Kernentwickler hat eine zwischen 19% und 32% grössere Chance, dass sein Fehler durch einen der Seinesgleichen behoben wird. Die Dauer der Projektbeteiligung hat einen negativen Einfluss. Pro Jahr der Projektbeteiligung verringert

sich die Chance um 2% - 5%.

Den grössten Effekt hat jedoch der Schweregrad des Fehlers. Releasekritische Fehler werden mit einer um 37% bzw. 27% grösseren Wahrscheinlichkeit von einem offiziellen Entwickler behoben. Diese kümmern sich insbesondere um diese sehr wichtigen Fehler. Auch sind diese Fehler zeitkritischer, da sie vor einem Release behoben werden müssen und die Benutzer stärker beeinträchtigen. Unter diesem Aspekt macht es Sinn, dass solche Fehler von offiziellen Entwicklern behoben werden, die das verbesserte Paket direkt ohne Umweg der Distribution hinzufügen können.

Die Konstante kann in diesem Modell als Basiswahrscheinlichkeit aufgefasst werden, dass ein Fehler von einem offiziellen Entwickler oder einem Kernentwickler behoben wird. Der hohe Wert zeigt, wie schon aus Tabelle 5.8 zu vermuten war, dass unabhängig von den anderen ins Modell einbezogenen Einflüssen die Chance, dass ein Fehler von einem dieser Entwickler behoben wird, sechs Mal grösser ist, als dass er von einem einfachen Entwickler oder einem Contributor behoben wird. Dieser Wert ist im reduzierten Datensatz sogar noch grösser. Im vollständigen Datensatz wurde zusätzlich noch die Art der Fehlerbehebung eingefügt. Da im reduzierten Datensatz nur Fehler enthalten sind, welche durch ein Softwarepaket behoben wurden, macht diese Variable dort keinen Sinn. Die Unterschiede sind dadurch zu erklären, dass nicht allen Personen alle Arten der Fehlerbehebung zugänglich sind. Nur Entwickler können ein verbessertes Softwarepaket hochladen und das Entfernen von Paketen aus der Distribution ist nur einem kleinen Kreis von Entwicklern gestattet.

5.4 Grenzen des Datensatzes

Die grösste Stärke des Datensatzes, dass er ausschliesslich aus Prozessdaten besteht, ist gleichzeitig auch seine grösste Schwäche. In den Prozessdaten finden sich keine näheren Angaben zu nicht direkt im Prozess der Open Source Entwicklung sich manifestierenden Eigenschaften der Personen. So fehlen sozio-ökonomische Indikatoren und direkte Informationen zur technischen Kompetenz. Das Fehlen dieser Informationen schränkt die Analyse in zweierlei Hinsicht ein: Erstens können einige sehr interessante Hypothesen mit dem Datensatz nicht untersucht werden. So lassen sich keine Aussagen über Reputationseffekte ausserhalb des Debian Projekts machen. Es ist jedoch eher unwahrscheinlich, dass die interne Reputationsnutzen, U_{repint} in der Nutzenfunktion (zur Nutzenfunktion siehe Abschnitt 3.2.1) als Motivation zur Beteiligung an einem Open Source Projekt alleine ausreichend ist. Ich gehe davon aus, dass der externe Nutzen einen weit grösseren Einfluss hat. Dieser lässt sich jedoch ohne sozio-ökonomische Indikatoren wie den Lohn nicht messen. Zweitens wären auch zusätzliche Indikatoren als Kontrollvariablen hilfreich. Zentral sind hier Informationen zur technischen Kompetenz des Fehlerberichterstatters.

Doch nicht nur der Fall, dass Reputation über das Debian Projekt hinaus Wirkung entfaltet, entzieht sich der Untersuchung. Auch der umgekehrte Fall, dass ausserhalb des Projekts erworbene Reputation innerhalb des Debian Projekts wirksam wird, wäre aus den theoretischen Überlegungen zu erwarten. Die Reputation hoch angesehener Entwickler aus anderen Open Source Projekten müsste auch innerhalb des Debian Projekts eine

Wirkung zeigen. Interessante wären in diesem Zusammenhang beispielsweise Effekte in Fehlerberichten, die von den ursprünglichen Autoren der in Debian enthaltenen Software stammen. So wäre zu erwarten, dass beispielsweise Linus Torvalds, der führende Entwickler des Linux Kernels, von einer grossen Reputation profitiert, obwohl er sich nicht am Debian Projekt beteiligt. Von ihm gemeldete Fehler müssten mit einer hohen Wahrscheinlichkeit in vergleichsweise kurzer Zeit behoben werden. Daten dazu fehlen jedoch im untersuchten Datensatz. Torvalds hat sich zwar auf Mailinglisten des Debian Projekts geäussert, jedoch findet sich im Datensatz kein Fehlerbericht von ihm.

Grenzen werden der Analyse jedoch nicht nur durch im Datensatz nicht verfügbare Informationen gesetzt, sondern auch durch Informationen, welche grundsätzlich in den Daten vorhanden wären, sich aber nicht automatisch erheben lassen. Hier sind insbesondere zusätzliche Informationen über die Art des Fehlers zu nennen. Die Schwierigkeit der Fehlerbehebung unterscheidet sich von Fehler zu Fehler sehr stark. Dies hat sicher einen Einfluss auf die Wahrscheinlichkeit und die Geschwindigkeit der Fehlerbehebung. Durch eine manuelle Klassifikation der Fehler liessen sich diese Informationen prinzipiell erheben. Dazu müsste jedoch zuerst ein Klassifikationssystem entwickelt und dieses danach zumindest auf ein ausreichendes Subsample angewandt werden. Der Aufwand dieses Unterfangens wäre enorm und hätte den Rahmen der vorliegenden Arbeit bei weitem gesprengt. Doch nicht nur die Eigenschaften des Fehlers haben einen Einfluss, sondern auch das Projektumfeld beinflusst die Fehlerbehebung. So spielt es beispielsweise eine Rolle, ob in kürze ein neues Release der Distribution veröffentlicht werden soll oder ob ein Fehler auch erst später behoben werden kann. Ich habe versucht, eine Dummy-Variable für ein bevorstehendes Release ins Modell einzuführen. Diese sehr einfache Modellierung hat jedoch keine Effekte gezeigt. Dies deutet aber wohl eher darauf hin, dass die Zusammenhänge komplexer sind und nicht durch eine einfache Dummy-Variable modelliert werden können.

6 Schlussbemerkungen

Empirisch zeigt sich in Open Source Projekten ein erstaunlich hohes Mass an Kooperation. Ohne das Zusammenwirken vieler Entwickler kann das Entstehen einer so vielfältigen Palette von Open Source Software nicht erklärt werden. Auch in meinen Daten zum Debian Projekt zeigt sich diese Kooperation. Ungefähr 90% aller Fehlerberichte werden behoben. Gerade bei den Fehlerberichten kann dies sicher nicht alleine mit dem Nutzen der Fehlerbehebung für den Fehlerbehebenden erklärt werden. Die Fehlerberichte beschreiben Probleme, von denen andere Nutzer betroffen sind.

Ausgangspunkt meiner Arbeit war die Frage, wie diese Kooperation unter Open Source Entwicklern erklärt werden kann. Open Source Entwicklung gleicht einem iterierten Gefangenendilemma. Gegenseitige Kooperation rationaler Entwickler kann in diesem Dilemma nur erklärt werden, wenn die Entwickler zwischen kooperierenden und defektierenden Akteuren diskriminieren können. Die einfachste Möglichkeit, wie Entwickler kooperationsbereite Personen von Free-Ridern unterscheiden können, ist eine wiederholte Interaktion. Sie erlaubt es den Entwicklern eine Tit-for-Tat Strategie anzuwenden. Meine Ergebnisse zeigen einige Hinweise, dass direkt reziprokes Kooperationsverhalten in der Open Source Entwicklung eine Rolle spielt. Die statistischen Modelle zeigen, dass vergangene Interaktion zu kooperativem Verhalten motiviert. Vergangene Interaktion kann hier relativ problemlos mit vergangener Kooperation gleichgesetzt werden, da Nicht-Kooperation primär die Absenz einer Interaktion ist. Die Entwickler verwenden eine Strategie, welche Kooperation in der Vergangenheit belohnt. Ob die Entwickler jedoch wirklich eine Tit-for-Tat Strategie anwenden, kann so nicht gezeigt werden. Interessant dazu wäre ein Datensatz, in welchem Defektion direkt beobachtbar ist. Auch wurden das Ausmass und die Art der vergangenen Interaktion nicht berücksichtigt. Mit dem vorhandenen Datensatz könnten aber noch wesentlich detailliertere Indikatoren zur vergangenen Interaktion gebildet werden. Spannend wäre beispielsweise herauszufinden, ob mehrfach wiederholte Interaktion einen Zusatzeffekt hat oder ob es Unterschiede zwischen den verschiedenen Form der Interaktion gibt. Damit könnte die Wirkungsweise der wiederholten Interaktion genauer erfasst werden.

Vergangene Interaktion kann aber, wie bereits im Theoriekapitel erwähnt, auch als Proxy dafür aufgefasst werden, dass Alter und Ego zusammen im gleichen Teilbereich des Projekts aktiv sind. Dann zeigen die Ergebnisse, dass mit Personen verstärkt kooperiert wird, welche aus einem Subprojekt näher bekannt sind. Diese Lesart widerspricht aber der zuerst präsentierten nicht. Sie ist mehr als Ergänzung dazu zu verstehen. Sie bestätigt, dass Faktoren die Kooperation begünstigen, welche die «anonyme» Zusammenarbeit in einem Open Source Projekt entanonymisieren.

Doch direkte Reziprozität reicht nicht aus, um Kooperation in einem Open Source Projekt zu erklären. Mit meiner sehr breiten Definition tritt wiederholte Interaktion

zwar recht häufig auf. Doch es ist keineswegs so, dass jeder jeden kennen würde. Deshalb braucht es Mechanismen, welche Kooperation auch unter Abwesenheit direkter Information stabilisieren. Im Theorieteil meiner Arbeit habe ich ausführlich dargelegt, dass Reputation als Signal für Kooperationsbereitschaft diese Aufgabe übernehmen kann. Aus direkter Reziprozität wird die über Reputationssignale regulierte indirekte Reziprozität. A kooperiert mit B, weil dieser in der Vergangenheit mit C kooperiert hat, was in Zukunft C wiederum dazu motiviert, mit A zu kooperieren, und so weiter. Die Ergebnisse zu den Reputationshypothesen sind etwas weniger eindeutig als diejenigen zur vergangenen Interaktion. Die Effekte auf die Wahrscheinlichkeit, dass Kooperation überhaupt stattfindet sind relativ klein. Sie zeigen sich nur im reduzierten Datensatz signifikant und deutlich. Die Medianzeiten zur Fehlerbehebung reduzieren sich je nach Status des Fehlerberichterstatters erheblich. Auch unter den verschiedenen Entwicklerstatus zeigen sich signifikante Unterschiede. Es lohnt sich also nicht nur, sich als Entwickler am Projekt zu beteiligen um Kooperationsbereitschaft zu signalisieren, sondern eine stärkere Beteiligung fördert die Kooperationsbereitschaft der anderen Entwickler weiter. Auch in den multivariaten Modellen zeigen sich die gleichen Reputationseffekte. Diese Modelle berücksichtigen sowohl Wahrscheinlichkeit als auch die Geschwindigkeit der Fehlerbehebung.

Dass Kooperation durch Reputation begünstigt wird, zeigt sich auch darin, dass Entwickler mit einer höheren Reputation vermehrt untereinander kooperieren. Je stärker sich ein Entwickler am Projekt beteiligt, desto höher ist die Chance, dass seine Fehler von einem Entwickler mit hoher Reputation behoben wird. Allerdings ist dieser Effekt nur im vollständigen Datensatz signifikant. Einzig ein Einfluss der Reputation auf die Anzahl der Personen, welche sich an der Behebung eines Fehler beteiligt, kann nicht bestätigt werden (Hypothese C4). Hier zeigt sich sogar ein umgekehrter Effekt. Die Reputation des Fehlerberichterstatters hat einen negativen Einfluss auf die Anzahl der beteiligten Personen. Die Gründe für diesen Effekt müssen, wie bereits erwähnt, noch genauer untersucht werden.

Reputation kann auch als eine Investition in die Zukunft betrachtet werden. Wie wir gesehen haben, lohnt sich der Aufbau einer guten Reputation. Die Vermutung liegt deshalb nahe, dass Entwickler nach dem Eintritt ins Projekt strategisch eine gute Reputation aufbauen. Haben sie sich jedoch einmal einen guten Ruf als kooperationswillige Entwickler gesichert, dann nimmt der Nutzen zusätzlicher Investitionen in die Reputation ab. Ich habe versucht, diesen Effekt mit dem durchschnittlichen Aktivitätsverlauf über die Zeit zu visualisieren. Die durchschnittliche Aktivität der Entwickler steigt tatsächlich in einer Anlaufphase an und nimmt danach im späteren Verlauf langsam wieder ab. Dies ist zumindest ein Hinweis, dass Open Source Entwickler anfänglich stärker in ihre Reputation investieren. Mit der rein graphischen Analyse kann jedoch über die Gründe für den Aktivitätsverlauf nur spekuliert werden. Eine genauere Analyse ist sicher notwendig, um zu beweisen, dass der Aufbau einer Reputation in der Anfangsphase und der «Konsum der Reputationsrendite» wirklich die zentralen Erklärungsfaktoren für den Verlauf der Aktivitätskurve sind. Teil einer solchen Analyse wäre auch, zuerst zu klären, wie die in dieser Arbeit nur aggregiert als Durchschnittswert für alle Entwickler dargestellten Aktivitätskurven auf der Individualebene überhaupt im Rahmen einer erklärenden Analyse verwendet werden können. Einschränkend muss auch festgehalten werden, dass diese

Hypothese nur an einem Teil der Entwickler geprüft wurde, welche über längere Zeit und relativ konstant aktiv sind.

Die Frage, wie ein Akteur zur Kooperation motiviert wird, ist nur die eine Seite der Medaille. Oft kommt als Interaktionspartner nicht nur ein Entwickler in Frage. Mehrere Entwickler kümmern sich um ein Softwarepaket und sind für die Behebung eines Fehlers zuständig. Gewisse Fehler werden einfach an eine Mailingliste mit einem relativ diffusen Empfängerkreis weitergeleitet. Neben dem Reputationseffekt wird im Theorieteil deshalb ausgehend vom Freiwilligendilemma auch ein Effekt der Verantwortungsdiffusion postuliert. Dieser Effekt kann in den Fehlerberichten klar bestätigt werden. Je mehr Personen potentiell in Frage kommen, um einen bestimmten Fehler zu beheben, desto geringer ist die Chance, dass dieser behoben wird. Dieser Effekt ist insbesondere bei Fehlerberichten mit einem grossen und nicht klar bestimmten Empfängerkreis ausgeprägt. Interessanterweise werden diese Fehler aber nicht langsamer behoben. Falls sie behoben werden, dann erfolgt dies vergleichsweise rasch.

Im N-Personen Gefangenendilemma, wie es im Theorieteil als Modell des Open Source Entwicklungsprozess entworfen wurde, ist Kooperation sehr fragil. Gleichwohl gelingt es auch sehr grossen Projekten wie Debian Kooperation über längere Zeit aufrechtzuerhalten. Dies lässt sich dadurch erklären, dass sich das Projekt in zahlreiche relativ kleine Subgruppen aufteilt. Diese Aufteilung ermöglicht es dem Projekt überhaupt erst so riesig zu werden. Durch die Aufteilung ist Kooperation auch nur mit einzelnen Akteuren oder Subgruppen möglich. Defektoren können so überhaupt erst von der Kooperation ausgeschlossen werden, ohne dass die Kooperation insgesamt zusammenbricht. Im Datensatz zeigt sich, dass die durchschnittliche Subgruppe klein ist. Diese Gruppen sind jedoch keine festen Einheiten, sondern bilden sich von Fall zu Fall und existieren oft auch nur für eine kurze Zeit. Jeder einzelne Entwickler ist in mehreren solchen Gruppen aktiv. Eine solche Subgruppe sind beispielsweise die Entwickler, welche sich gemeinsam um ein einzelnes Softwarepaket kümmern oder welche zusammen einen Fehler beheben. Die durchschnittliche Anzahl der Paketbetreuer der einzelnen Fehler liegt bei 1.8. Der Median ist sogar 1. Das heisst mehr als die Hälfte der Fehler gehören zu Paketen, welche von einer einzigen Person betreut werden.[1] Der Mittelwert der Personen, welche sich an einem Fehler beteiligen ist 2.5. Der Median liegt bei 2. In vielen Fällen reduziert sich damit das N-Personen Dilemma auf ein 2-Personen Spiel.

Einige interessante Richtungen, in die weiter geforscht werden kann, ergeben sich bereits aus den in Abschnitt 5.4 analysierten Defiziten des Datensatzes. Eine relativ einfache Variante ist die Prüfung der Hypothesen an einem Subsample mit sehr homogenen Fehlerberichten. Diese müssten wohl manuell ausgewählt werden. Dadurch sollte sich insbesondere die erklärte Varianz steigern lassen. Für weitere Forschungen wäre es wünschenswert, die Prozessdaten des Datensatzes um Informationen zu den Entwicklern zu erweitern. Diese können über eine direkte Befragung der Entwickler erhoben werden. Dabei müssen zwei zentrale Probleme gelöst werden. Erstens sind die Entwickler nur über Email erreichbar. Eine herkömmliche Befragung auf dem Papierweg kommt nicht in

[1] Dies Zahlen sind nur als Illustration zu verstehen. Sie beziehen sich auf den Datensatz der Fehlerberichte. Die Anzahl der Paketbetreuer von Paketen mit vielen Fehlern sind deshalb stärker gewichtet.

Frage, da keine Wohnadressen bekannt sind. Zweitens ist eine solche Befragung insbesondere dann interessant, wenn die Antworten auf der Ebene des Individuums mit den Prozessdaten verknüpft werden können. Damit dies möglich ist, kann jedoch die sonst übliche Anonymität der Antworten zumindest gegenüber dem Forscher nicht gewährleistet werden. Es ist aber unsicher, ob ohne Anonymität zu den besonders interessanten Fragen, beispielsweise zum Einkommen, ein genügend grosser Rücklauf erreicht werden kann.

Mit Hilfe einer direkten Befragung der Entwickler könnten auch die Reputationskonstrukte validiert werden. Eine solche Validierung wäre zur Absicherung der hier vorgelegten Ergebnisse sicher wertvoll. Wie eine solche Validierung durchgeführt werden kann, müsste aber noch im Detail geklärt werden. Eine Möglichkeit wäre die Befragten die Reputation einer Auswahl von Entwicklern direkt bewerten zu lassen.

Die mit dem vorhandenen Datensatz bearbeitbaren soziologischen Fragestellungen sind mit der vorliegenden Arbeit mit Sicherheit noch nicht erschöpft. Der grosse Teil der Daten, welche sich auf die Aktivität auf den Mailinglisten bezieht, wurde nur am Rande in die Analyse miteinbezogen. Dort können sicherlich noch zahlreiche spannende Hypothesen geprüft werden. Interessant wäre beispielsweise, ob sich auch auf den Mailinglisten Reputationseffekte nachweisen lassen. Ob also Mails von Personen mit einer hohen Reputation schneller oder von mehr Personen beantwortet werden. Soziologische Untersuchungen zum Sozialsystem «Mailingliste» sind noch nicht sehr zahlreich. Anschlussfähig für diese Fragestellung wären allenfalls die Studien von Stegbauer (2001), der mehrere Mailinglisten unter netzwerkanalytischen Gesichtspunkten betrachtet hat.

Insbesondere die Netzwerkperspektive bietet noch zahlreiche spannende Fragestellungen. Das Debian Projekt kann als soziales Netzwerk aufgefasst werden. Die Verbindungen in diesem Netzwerk werden durch die gemeinsame Arbeit an Softwarepaketen, durch das gemeinsame beheben eines Fehlers oder durch die gemeinsame Beteiligung an einer Diskussion auf einer Mailingliste gebildet. Crowston & Howison (2005) haben bereits erste netzwerkanalytische Untersuchungen mit einem Datensatz der Open Source Plattform sourceforge.net unternommen. Ihr Ansatz kann auf das Debian Projekt adaptiert werden. Sie zeigen, dass die Netzwerkstruktur je nach Projekt sehr unterschiedlich ist. Es gibt sowohl sehr zentralisierte, wie auch stark dezentralisierte Netzwerke. Für das Debian Projekt wäre wohl eher eine dezentrale Struktur zu erwarten. Rudimentäre Netzwerkindikatoren wurden auch für die Fehlerberichte in die Analyse einbezogen. Diese zeigten jedoch keine signifikanten Effekte. Hier bieten sich aber viele Möglichkeiten für ausführlichere Betrachtungen. In erster Linie müsste wohl die Verteilung des Netzwerks betrachtet werden. Die im deskriptiven Teil beschriebene Verteilung der Projektarbeit lässt vermuten, dass die Verteilung derjenigen eines «Scale-free» Netzwerkes nahe kommt. Im Netzwerk könnte auch geprüft werden, ob sich Subgruppen empirisch identifizieren lassen. Beispielsweise Personen, welche an verwandten Softwarepaketen arbeiten.

Mit den Daten lässt sich auch die zeitliche Entwicklung des Netzwerkes rekonstruieren. Damit eignen sie sich nicht nur zur statischen Netzwerkanalyse, sondern es kann auch die Dynamik der Projektentwicklung in die Analyse miteinbezogen werden. Das grösste Problem einer solchen dynamischen Betrachtung ist, wie die hohe Komplexität der Analyse sinnvoll bewältigt werden kann. Im Modell von Bitzer & Schröder (2005) werden einige Hypothesen zur dynamischen Entwicklung von Open Source Projekten aus ökonomischer

Perspektive formuliert. Diese können allenfalls als Grundlage zur Hypothesenbildung beigezogen werden. Sie vermuten, dass sowohl der Start, wie auch die Kollaps eines Open Source Projekts Mitläufereffekte («bandwagon dynamics») zeigt. Sie formulieren auch konkrete Hypothesen dazu, welche Art von Entwickler sich beteiligen wird und wann sie free-riden (Bitzer & Schröder 2005: 402).

Interessant wäre auch, zu prüfen, ob ein Substitutionseffekt zwischen Reputation und wiederholter Interaktion besteht. Wehrli (2005) hat einen solchen Effekt auf eBay.de nachgewiesen. Falls sich Entwickler bereits aus vergangenen Interaktionen kennen, sinkt der Wert des Reputationssignals. Sie müssen sich nicht mehr auf die im Allgemeinen weniger zuverlässige Reputationsinformation verlassen, sondern verfügen über zuverlässigere Informationen über die Kooperationsbereitschaft aus vergangenen Interaktionen. Statistisch würde sich ein solcher Effekt über einen negativen Einflusses des Interaktionsterms der Reputationsvariablen mit der Variablen für vergangene Interaktion zeigen. Wie genau sich eine solche Analyse aber durchführen liesse, müsste noch geklärt werden. Insbesondere weil die Information über die vergangene Interaktion nur im Falle eines behobenen Fehlers verfügbar ist und sich in den entsprechenden zugehörigen Modellen kein Reputationseffekt nachweisen lässt.

Die Fehlerdatenbank des Debian Projekts ist nicht die einzige, für welche alle Daten öffentlich zugänglich sind. Praktisch jedes grössere Open Source Projekt hat eine öffentliche Fehlerdatenbank. An diesen kann eine Reproduktion der hier vorgelegten Ergebnisse unternommen werden. Um meine Ergebnisse besser abzusichern, wäre dies sicher ein sinnvoller nächster Schritt. So könnten auch eventuelle Unterschiede zwischen den Projekten aufgedeckt werden.

Es soll nicht unerwähnt bleiben, dass das Phänomen der Open Source Software Entwicklung auch zahlreiche Möglichkeiten für qualitative soziologische Untersuchungen bereit hält. So wurden die in dieser Arbeit verwendete Einteilung der Entwickler in verschiedene Kategorien stark Ad-Hoc ohne fundierte theoretische Grundlage gebildet. Hier wäre die Bildung einer fundierten Typologie unbedingt notwendig. Auch würde eine qualitative Untersuchung die Möglichkeit eröffnen, zu prüfen, ob die unterstellten Reputationsüberlegungen auch in den subjektiven Konzepten der Entwickler präsent sind. Auch die Motivation der Open Source Entwickler, sich an einem Projekt zu beteiligen kann mit qualitativen Methoden gut untersucht werden.

Zur Motivation von Open Source Entwicklern ist schon viel publiziert worden. Diese Publikationen basieren aber teilweise mehr auf Spekulationen und Eindrücken einiger weniger Entwickler, denn auf sozialwissenschaftlich empirischen Fakten (Grassmuck 2002, Raymond 2001: beispielsweise). Auch die bisher durchgeführten Online Befragungen zu diesem Thema genügend mit wenigen Ausnahmen nicht strengen sozialwissenschaftlichen Kriterien. Die genauere Untersuchung der Motivation, sich an einem Open Source Projekt zu beteiligen, wäre aber notwendig, um die in dieser Arbeit skizzierte Nutzenfunktion zu verifizieren und detaillierter zu beschreiben. Interessant dazu ist insbesondere die Arbeit von Lakhani & Wolf (2005). Sie finden heraus, dass insbesondere intrinsische auf Kreativität bezogene Faktoren eine grosse Rolle für die Motivation spielen. Erstaunlich ist insbesondere, dass sich in ihrer Untersuchung kein negativer Zusammenhang zwischen extrinsischer Motivation über monetäre Entschädigungen und intrinsischer Motivation besteht, wie er in anderen Zusammenhängen als der Open Source Entwicklung beobachtbar ist.

Literaturverzeichnis

Axelrod, R. (1984), *The Evolution of Cooperation*, Basic Books.

Bitzer, J. & Schröder, P. J. (2005), 'Bug-fixing and code-writing: The private provision of open source software', *Information Economics and Policy* **17**(3), 389–406.

Blossfeld, H. & Rohwer, G. (1995), *Techniques of event history modeling: new approaches to causal analysis*, Lawrence Erlbaum Associates.

Brand, A. (n.d.*a*), Die Struktur, Eintritt, Leistungserstellung, Motivation und Kontrolle in einem Open Source-Projekt.
URL: `http://www.kde.de/nachrichten/detail/Paper_KDE_Andreas_Brand.pdf, 22.12.2008`

Brand, A. (n.d.*b*), Fallstudie horizontales elektronisches Arbeitsnetz/Open Source-Projekt.
URL: `http://www.soz.uni-frankfurt.de/arbeitslehre/pelm/docs/FALLSTUDIE%20Open%20Source%20V1.pdf,23.7.2007`

Brand, A. & Holtgrewe, U. (2004), KDE im Kontext: Open Source Software Entwicklung und öffentliche Güter. erscheint in Moldaschl, Manfred/Weber Hajo (Hg.): Wissen und Innovation - Beträge zur Ökonomie der Wissensgesellschaft.
URL: `http://www.uni-due.de/imperia/md/content/soziologie/abuh-indsoz-604.pdf,21.12.2008`

Brooks, F. (1975), *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley.

Cameron, A. C. & Trivedi, P. K. (1998), *Regression Analysis of Count Data*, Cambridge University Press, Cambridge.

Coleman, G. E. (2005), The Social Construction of Freedom in Free and Open Source Software: Hackers, Ethics, and the Liberal Tradition, PhD thesis, Deptartment of Anthropology, University of Chicago.

Coleman, G. & Hill, B. (2005), The Social Production of Ethics in Debian and Free Software Communities. Anthropological Lessons for Vocational Ethics, *in* S. Koch, ed., 'Free/open Source Software Development', Idea Group Pub.

Cox, D. R. (1972), 'Regression models and life-tables', *Journal of the Royal Statistical Society. Series B (Methodological)* **34**(2), 187–220.

Crowston, K. & Howison, J. (2005), The social structure of Free and Open Source software development.

Dasgupta, P. (1988), Trust as a Commodity, *in* D. Gambetta, ed., 'Trust: Making and Breaking Cooperative Relations', Basil Blackwell, Oxford, pp. 49–72.

David, P. A., Waterman, A. & Arora, S. (2003), FLOSS-US. The Free/Libre/Open Source Software Survey for 2003.
URL: `http://www.stanford.edu/group/floss-us/report/FLOSS-US-Report.pdf, 22.12.2008`

Diekmann, A. (1985), 'Volunteer's Dilemma', *The Journal of Conflict Resolution* **29**(4), 605–610.

Diekmann, A. (1993*a*), 'Cooperation in an Asymetric Volunteer's Dilemma Game. Theory and Experimental Evidence', *International Journal of Game Theory* **22**, 75–85.

Diekmann, A. (1993*b*), 'Sozialkapital und das Kooperationsproblem in sozialen Dilemmata', *Analyse & Kritik* **15**, 22–35.

Diekmann, A. (2004), 'The Power of Reciprocity. Fairness, Reciprocity, and Stakes in Variants of the Dictator Game', *Journal of Conflict Resolution* **48**(4), 487–505.

Diekmann, A. & Manhart, K. (1989), 'Kooperative Strategien im Gefangenendilemma. Comnputersimulation eines N-Personen-Spiels', *Analyse & Kritik* **11**, 134–153.

Diekmann, A. & Mitter, P. (1984), *Methoden zur Analyse von Zeitverläufen. Anwendungen stochastischer Prozesse bei der Untersuchung von Ereignisdaten*, B.G.Teubner, Stuttgart.

Diekmann, A. & Voss, T. (2004), *Rational Choice Theorie in den Sozialwissenschaften: Anwendungen und Probleme*, Oldenbourg Wissenschaftsverlag, München.

Feller, J., Fitzgerald, B., Hissam, S. A. & Lakahni, K. R., eds (2005), *Perspectives on Free and Open Source Software*, MIT Press.

Friedman, J. W. (1971), 'A non-cooperative equilibrium for supergames', *Review of Economic Studies* **38**(113), 1–12.

Garzarelli, G. & Galoppini, R. (2003), Capability Coordination in Modular Organization: Voluntary FS/OSS Production and the Case of Debian GNU/Linux.
URL: `http://opensource.mit.edu/papers/garzarelligaloppini.pdf,27.1.2009`

Ghosh, R., Glott, R., Krieger, B. & Robles, G. (2002), Free/Libre and Open Source Software: Survey and Study. Survey of Developers.
URL: `http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf,22.12.2008`

Ghosh, R. & Ved Prakash, V. (2000), 'The Orbiten free software survey', *First Monday* **5**(7).
URL: `http://firstmonday.org/`

Grambsch, P. M. & Therneau, T. M. (1994), 'Proportional Hazards Tests and Diagnostics Based on Weighted Residuals', *Biometrika* **81**(3), 515–526.

Granovetter, M. (1985), 'Economic Action and Social Structure: The Problem of Embeddedness', *American Journal of Sociology* **91**(3), 481–510.

Grassmuck, V. (2002), *Freie Software. Zwischen Privat- und Gemeineigentum*, Bundeszentrale für politische Bildung, Bonn.

Hamburger, H. (1973), 'N-person prisoner's dilemma', *Journal of Mathematical Sociology* **3**(1), 27–48.

Hardin, G. (1968), 'The tragedy of the commons', *Science* **162**(3859), 1243–1248.

Hardin, R. (1971), 'Collective Action as an Agreeable N-Prisoners' Dilemma', *Behavioral Science* **16**(5), 472–481.

Harrell, F. E. (2008), *Design: Design Package*. R package version 2.1-2.
URL: `http://biostat.mc.vanderbilt.edu/s/Design,http://biostat.mc.vanderbilt.edu/rms`

Hars, A. & Ou, S. (2002), 'Working for Free? Motivations for Participating in Open-Source Projects', *International Journal of Electronic Commerce* **6**(3), 25–39.

Hertel, G., Niedner, S. & 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.

Hosmer, D., Lemeshow, S. & May, S. (1999), *Applied Survival Analysis: Regression Modeling of Time to Event Data*, John Wiley & Sons.

Johnson, J. P. (2002), 'Open Source Software: private provision of a public good', *Journal of Economics & Management Strategy* **11**(4), 637–662.

Koch, S. (2005), 'Effort Modeling and Programmer Participation in Open Source Software Projects', *Working Papers on Information Systems, Information Business and Operations* .

Koch, S. & Schneider, G. (2000), 'Results from Software Engineering Research into Open Source Development Projects Using Public Data', *Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft* **22**.

Krishnamurthy, S. (2002), 'Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects', *First Monday* **7**(6).
URL: `http://firstmonday.org/`

Lakhani, K. R. & Wolf, R. G. (2005), Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, *in* J. Feller, B. Fitzgerald, S. Hissam & K. R. Lakhani, eds, 'Perspectives on Free and Open Source Software', MIT Press, pp. 3–22.

Lerner, J. & Tirole, J. (2005), Economic Perspectives on Open Source, *in* J. Feller, B. Fitzgerald, S. Hissam & K. R. Lakhani, eds, 'Perspectives on Free and Open Source Software', MIT Press, pp. 47–78.

Levy, S. (1994), *Hackers. Heroes of the Computer Revolution*, Dell Publishing.

Long, J. S. (1997), *Regression models for categorical and limited dependent variables*, SAGE Publications Inc., Thousand Oaks.

Luce, D. R. & Raiffa, H. (1957), *Games and Decision. Introduction and Critical Survey*, Dover Publications, New York.

Mockus, A., Fielding, R. T. & Herbsleb, J. D. (2002), 'Two Case Studies of Open Source Software Development: Apache and Mozilla', *ACM Transaction on Software Engineering and Methodology* **11**(2), 309–346.

Mui, L., Mohtashemi, M. & Verma, S. (2004), A Group and Reputation Model for the Emergence of Voluntarism in Open Source Development. Publikation ?

Nowak, M. A. & Sigmund, K. (1998*a*), 'Evolutionof indirect reciprocity by image scoring', *NATURE* **393**, 573.

Nowak, M. A. & Sigmund, K. (1998*b*), 'The Dynamics of Indirect Reciprocity', *Journal of Theoretical Biology* **194**(4), 561–574.

Olson, M. (1965), *The Logic of Collective Action: Public Goods and the Theory of Groups*, Harvard University Press, Cambridge, MA.

O'Mahony, S. (2003), 'Guarding the commons: how community managed software projects protect their work', *Research Policy* **32**(7), 1179–1198.

O'Mahony, S. & Ferraro, F. (2004), Hacking Alone? The Effects of Online and Offline Participation on Open Source Community Leadership.

R Development Core Team (2008), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
URL: `http://www.R-project.org`

Rapoport, A. & Chammah, A. (1965), *Prisoner's Dilemma. A Study of Conflict and Co-operation.*, University of Michigan Press.

Raub, W. (1988), 'Problematic social situations and the "large number dilemma": A game-theoretical analysis.', *Journal of Mathematical Sociology* **13**(4), 311–357.

Raub, W. & Weesie, J. (1990), 'Reputation and efficiency in social interactions: An example of network effects', *American Journal of Sociology* **96**(3), 626–654.

Raymond, E. S. (2001), *The Cathedral and the Bazaar. Musings on Linux and Open Source ba an Accidental Revolutionary*, O'Reilly, Sebastopol, CA.

Schoenfeld, D. (1982), 'Partial residuals for the proportional hazards regression model', *Biometrika* **69**(1), 239–241.

Stegbauer, C. (2001), *Grenzen virtueller Gemeinschaft. Strukturen internetbasierter Kommunikationsforen*, Westdeutscher Verlag GmbH, Wiesbaden.

Stewart, D. (2005), 'Social Status in an Open-Source Community', *American Sociological Review* **70**, 823–842. Auswertung der Daten von advogato.

Suzuki, S. & Akiyama, E. (2005), 'Reputation and the evolution of cooperation in sizable groups', *Proceedings of the Royal Society B: Biological Sciences* **272**(1570), 1373–1377.

Therneau, T. & Lumley, T. (2008), *survival: Survival analysis, including penalised likelihood.* R package version 2.34-1.

Torvalds, L. & Diamond, D. (2001), *Just for fun: the story of an accidental revolutionary*, Harper Collins, New York.

Tutz, G. (2000), *Die Analyse kategorialer Daten. Anwendungsorientierte Einführung in Logit-Modellierung und kategoriale Regression*, Oldenbourg Wissenschaftsverlag, München.

Varian, H. R. (1992), *Microeconomic Analysis*, W. W. Norton & Company.

von Hippel, E. & von Krogh, G. (2003), 'Open Source Software and the „Private-Collective" Innovation Model: Issues for Organization Science', *Organization Science* **14**(2), 209–223.

Wedekind, C. & Milinski, M. (2000), 'Cooperation Through Image Scoring in Humans', *Science* **288**(5467), 850.

Wehrli, S. (2005), „Alles bestens, gerne wieder". Reputation und Reziprozität in Online-Auktionen, Master's thesis, Institut für Soziologie, Universität Bern.

Williams, S. (2002), *Free as in Freedom. Richard Stallman's Crusade for Free Software*, O'Reilly.

Wilson, R. (1985), Reputations in Games and Markets, *in* A. E. Roth, ed., 'Game-theoretic models of bargaining', Cambridge University Press, pp. 27–62.

A Ergänzende Statistiken

A.1 Deskriptive Statistik zum Datensatz Fehlerberichte

A.1.1 vollständiger Datensatz

Tabelle A.1: Deskriptive Statistik Entwicklerstatus (vollständiger Datensatz)

	Contrib.	einf. Entw.	off. Entw.	Kernentw.	miss
Fehlerberichterstatter	170799	31923	33841	67231	1548
	56%	11%	11%	22%	
Fehlerbeheber	13753	27478	60078	138930	65103
	6%	11%	25%	58%	

Anmerkung: N = 305342 (inkl. missing values); Prozentzahlen beziehen sich auf das Total ohne die missing values.

Tabelle A.2: Deskriptive Statistik Schweregrad (vollständiger Datensatz)

Schweregrad	**Anzahl**	**%**
wishlist	58583	19
minor	34128	11
normal	127057	42
important	44162	14
serious	25543	8
grave	13990	5
critical	1879	1

Anmerkung: N = 305342 (keine missing values)

Tabelle A.3: Deskriptive Statistik zum vollständigen Datensatz Fehlerberichte (metrische und Zählvariablen)

Variable	Me	$\bar{x}$	sd	Min	Max	miss
Variablen Fehlerberichterstatter						
eingesandte Fehlerberichte	57.00	299.76	613.38	0.00	5181.00	0
ln(einges. Fehlerberichte+1)	4.06	3.93	2.18	0.00	8.55	0
behobene Fehler	3.00	228.24	710.88	0.00	7167.00	0
ln(behobene Fehler+1)	1.39	2.40	2.63	0.00	8.88	0
Patches	0.00	9.40	27.91	0.00	302.00	0
ln(Patches+1)	0.00	0.93	1.41	0.00	5.71	0
Worte auf Mailinglisten	5632.00	64919.56	237031.15	0.00	5034571.00	0
ln(W. auf Mailinglisten+1)	8.64	7.43	4.20	0.00	15.43	0
Paketarbeit	0.00	362.75	1300.94	0.00	16725.00	0
ln(Paketarbeit+1)	0.00	2.38	2.94	0.00	9.72	0
Aktivität	1.31	1.62	1.20	0.00	4.63	0
Zus'arbeit Fehlerberichte	94.00	372.91	713.65	0.00	7027.00	0
ln(Zus'arbeit Fehlerberichte+1)	4.55	4.28	2.22	0.00	8.86	0
Zus'arbeit Softwarepakete	0.00	34.02	84.57	0.00	727.00	0
ln(Zus'arbeit Softwarepakete+1)	0.00	1.46	1.97	0.00	6.59	0
Projektbeteiligung (Jahre)	3.21	3.50	2.47	0.00	12.62	0
Variablen Fehlerbeheber						
eingesandte Fehlerberichte	102.00	313.78	592.06	0.00	5182.00	63681
ln(einges. Fehlerberichte+1)	4.63	4.56	1.69	0.00	8.55	63681
behobene Fehler	242.00	572.69	873.97	0.00	7163.00	63681
ln(behobene Fehler+1)	5.49	5.18	1.92	0.00	8.88	63681
Patches	3.00	13.34	28.96	0.00	300.00	63681
ln(Patches+1)	1.39	1.52	1.45	0.00	5.71	63681
Worte auf Mailinglisten	26590.00	122603.14	314835.22	0.00	4362513.00	63681
ln(W. auf Mailinglisten+1)	10.19	9.83	2.62	0.00	15.29	63681
Paketarbeit	395.00	908.86	1642.80	0.00	16735.00	63681
ln(Paketarbeit+1)	5.98	5.60	2.02	0.00	9.73	63681
Aktivität	2.67	2.60	0.90	0.00	4.64	63681
Zus'arbeit Fehlerberichte	424.00	709.70	834.86	0.00	7019.00	63681
ln(Zus'arbeit Fehlerberichte+1)	6.05	5.77	1.60	0.00	8.86	63681
Zus'arbeit Softwarepakete	30.00	64.33	95.24	0.00	727.00	63681
ln(Zus'arbeit Softwarepakete+1)	3.43	3.19	1.61	0.00	6.59	63681
Projektbeteiligung (Jahre)	4.40	4.56	2.37	-5.18	12.59	63681
Medianzeit Fehlerbehebung	0.73	1.42	2.40	0.00	63.12	74221
gemeinsame Variablen						
vergangene Interaktion	0.00	0.42	0.49	0.00	1.00	63681
Variablen Fehlerbericht						
Überlebenszeit (Tage)	56.56	264.50	432.80	0.00	2255.25	0
Überl. (nicht zensierte)	27.38	139.91	261.98	0.00	2208.16	0
Überl. (zensierte)	583.93	737.27	596.06	0.00	2255.25	0
Zensiert (0=Ereignis,1=zensiert)	0.00	0.21	0.41	0.00	1.00	0
Anzahl Paketbetreuer	1.00	1.80	1.60	1.00	18.00	93430
Worte im Fehlerbericht	138.00	372.03	5249.27	2.00	1351181.00	0
ln(Worte im Fehlerbericht+1)	4.93	4.94	0.98	1.10	14.12	0
Anz. beteiligte Personen	2.00	2.52	1.20	1.00	30.00	0

Anmerkung: N = 305342 (inkl. missing values); Me: Median, $\bar{x}$: Mittelwert, sd: Standardabweichung, Min: kleinster Wert, Max: grösster Wert, miss: missing values; negative Werte in der Projektbeteiligung der Fehlerbehebenden: Fehlerbehebender kommt erst ins Projekt, als der Fehler schon gemeldet wurde; Aktivität berechnet sich aus der Summe der logarithmierten und danach normierten Werte von eingesandte Fehlerberichte, behobene Fehler, Patches, Worte auf Mailinglisten und Paketarbeit

Tabelle A.4: Deskriptive Statistik Art der Fehlerbehebung (vollständiger Datensatz)

Art	**Anzahl**	**%**
verbessertes Softwarepaket	152972	63
Mail	81881	34
«wontfix»	3853	2
Entfernung aus Debian	2955	1
missing	63681	

Anmerkung: N = 305342 (inkl. missing values); Missing sind alle zensierten Fehlerberichte; Prozentzahlen beziehen sich auf das Total ohne die missing values.

A.1.2 reduzierter Datensatz

Tabelle A.5: Deskriptive Statistik Entwicklerstatus (reduzierter Datensatz)

	Contrib.	einf. Entw.	off. Entw.	Kernentw.	miss
Fehlerberichterstatter	67496	15842	15132	32314	598
	52%	12%	12%	25%	
Fehlerbeheber	1342	9546	25549	52748	42197
	2%	11%	29%	59%	

Anmerkung: N = 131382 (inkl. missing values); Prozentzahlen beziehen sich auf das Total ohne die missing values.

Tabelle A.6: Deskriptive Statistik Schweregrad (reduzierter Datensatz)

Schweregrad	**Anzahl**	**%**
wishlist	29694	23
minor	18851	14
normal	54406	41
important	15900	12
serious	9094	7
grave	3135	2
critical	302	0

Anmerkung: N = 131382 (keine missing values)

Tabelle A.7: Deskriptive Statistik zum reduzierten Datensatz Fehlerberichte (metrische und Zählvariablen)

Variable	Me	$\bar{x}$	sd	Min	Max	miss
Variablen Fehlerberichterstatter						
eingesandte Fehlerberichte	80.00	349.89	660.83	0.00	5181.00	0
ln(einges. Fehlerberichte+1)	4.39	4.23	2.13	0.00	8.55	0
behobene Fehler	7.00	253.69	748.23	0.00	7167.00	0
ln(behobene Fehler+1)	2.08	2.65	2.65	0.00	8.88	0
Patches	0.00	10.45	29.27	0.00	302.00	0
ln(Patches+1)	0.00	1.04	1.44	0.00	5.71	0
Worte auf Mailinglisten	7194.00	72396.41	248306.24	0.00	5034571.00	0
ln(W. auf Mailinglisten+1)	8.88	7.83	4.04	0.00	15.43	0
Paketarbeit	0.00	409.91	1396.78	0.00	16725.00	0
ln(Paketarbeit+1)	0.00	2.64	2.99	0.00	9.72	0
Aktivität	1.59	1.75	1.20	0.00	4.63	0
Zus'arbeit Fehlerberichte	133.00	417.83	748.50	0.00	7027.00	0
ln(Zus'arbeit Fehlerberichte+1)	4.90	4.56	2.13	0.00	8.86	0
Zus'arbeit Softwarepakete	0.00	38.76	90.56	0.00	727.00	0
ln(Zus'arbeit Softwarepakete+1)	0.00	1.64	2.03	0.00	6.59	0
Projektbeteiligung (Jahre)	3.54	3.77	2.50	0.00	12.60	0
Variablen Fehlerbeheber						
eingesandte Fehlerberichte	89.00	262.99	534.79	0.00	5182.00	41465
ln(einges. Fehlerberichte+1)	4.50	4.46	1.55	0.00	8.55	41465
behobene Fehler	212.00	477.82	733.31	0.00	7159.00	41465
ln(behobene Fehler+1)	5.36	5.18	1.64	0.00	8.88	41465
Patches	2.00	10.48	24.13	0.00	300.00	41465
ln(Patches+1)	1.10	1.33	1.38	0.00	5.71	41465
Worte auf Mailinglisten	22006.00	105954.41	292680.53	0.00	4310572.00	41465
ln(W. auf Mailinglisten+1)	10.00	9.81	2.24	0.00	15.28	41465
Paketarbeit	403.00	871.22	1512.04	0.00	16735.00	41465
ln(Paketarbeit+1)	6.00	5.80	1.64	0.00	9.73	41465
Aktivität	2.58	2.58	0.78	0.00	4.64	41465
Zus'arbeit Fehlerberichte	350.00	584.29	679.00	0.00	7018.00	41465
ln(Zus'arbeit Fehlerberichte+1)	5.86	5.70	1.36	0.00	8.86	41465
Zus'arbeit Softwarepakete	27.00	58.05	90.48	0.00	724.00	41465
ln(Zus'arbeit Softwarepakete+1)	3.33	3.17	1.47	0.00	6.59	41465
Projektbeteiligung (Jahre)	4.57	4.75	2.36	-3.86	12.37	41465
Medianzeit Fehlerbehebung	0.69	1.15	1.89	0.00	48.80	42809
gemeinsame Variablen						
vergangene Interaktion	0.00	0.46	0.50	0.00	1.00	41465
Variablen Fehlerbericht						
Überlebenszeit (Tage)	43.68	240.77	417.22	0.00	2254.90	0
Überl. (nicht zensierte)	14.62	69.67	147.36	0.00	2145.70	0
Überl. (zensierte)	459.84	611.79	550.76	0.00	2254.90	0
Zensiert (0=Ereignis,1=zensiert)	0.00	0.32	0.46	0.00	1.00	0
Anzahl Paketbetreuer	1.00	1.63	1.32	1.00	18.00	33745
Worte im Fehlerbericht	131.00	344.80	6424.96	2.00	1351181.00	0
ln(Worte im Fehlerbericht+1)	4.88	4.87	0.96	1.10	14.12	0
Anz. beteiligte Personen	2.00	1.82	0.38	1.00	2.00	0

Anmerkung: N = 131382 (inkl. missing values); Me: Median, $\bar{x}$: Mittelwert, sd: Standardabweichung, Min: kleinster Wert, Max: grösster Wert, miss: missing values; negative Werte in der Projektbeteiligung der Fehlerbehebenden: Fehlerbehebender kommt erst ins Projekt, als der Fehler schon gemeldet wurde; Aktivität berechnet sich aus der Summe der logarithmierten und danach normierten Werte von eingesandte Fehlerberichte, behobene Fehler, Patches, Worte auf Mailinglisten und Paketarbeit

A.2 Parametrisches Modell zur Fehlerbehebung

Tabelle A.8: Effekte auf die Fehlerbehebungsrate (Weibull-Regression)

	HT		alle	reduziert	alle	reduziert
Variablen Fehlerberichterstatter						
Kategorie (Referenz Contributor)						
einfacher Entwickler	C2	+	-0.32***	-0.51***	-0.15***	-0.21***
offizieller Entwickler	C2	+	-0.34***	-0.86**	0.01	-0.07
Kernentwickler	C2	+	-0.49***	-1.09***	-0.03	-0.14*
Dauer der Projektbeteiligung		+	0.05***	0.14***	-0.01*	-0.01
Variablen Fehlerbeheber						
Kategorie (Referenz Contributor)						
einfacher Entwickler		+			-0.89***	-1.86***
offizieller Entwickler		+			-0.86***	-1.91***
Kernentwickler		+			-0.98***	-2.06***
Dauer der Projektbeteiligung		+			-0.04***	0.03***
Medianzeit Fehlerbehebung		–			0.10***	0.07***
gemeinsame Variablen						
vergangene Interaktion	A1	+			-0.13***	-0.15***
Variablen Fehlerbericht						
Releasekritisch		+	-1.96***	-2.99***	-1.23***	-1.90***
Anzahl Paketbetreuer	E1	–	0.08***	0.28***	-0.01***	-0.02*
Worte im Fehlerbericht		+	0.19***	0.23***	0.07***	-0.04*
Art der Fehlerbehebung (Referenz Softwarepaket)						
Mail					0.41***	
«Wontfix»					0.36***	
Entfernung					1.49***	
Modellparameter						
Konstante			4.29***	3.87***	4.77***	5.40***
$ln(\alpha)$			0.91***	0.96***	0.68***	0.61***
N			210'791	97'184	168'460	69'442
Events			169'589	70'057	168'460	69'442
missing			94'551	34'198	136'882	61'940
Clusters			20'642	12'038	17'923	9'068

Anmerkungen: Weibull-Regression mit abhängiger Variable Zeit bis zur Fehlerbehebung; Maximum-Likelihood-Schätzung der Effekte auf die Hazardrate; Robuste Standardfehler mit Clustering für Fehlerberichterstatter (aus Platzgründen nicht ausgewiesen); Signifikanzniveaus: * p<=0.05, ** p<=0.01, *** p<=0.001; Deskriptive Statistiken zu den Variablen im Anhang A.1; Im Weibull-Modell wird eine monotone fallende oder steigende Hazardrate angenommen. Im Unterschied zur Cox-Regression zeigen negative Koeffizienten einen beschleunigenden Einfluss an. $ln(\alpha)$ ist ein zusätzlicher geschätzter Parameter für die Steigung der Hazardrate. Ein positiver Wert zeigt eine steigende Hazardrate an.

B Selbständigkeitserklärung

Ich erkläre hiermit, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetzes vom 5. September 1996 über die Universität zum Entzug des aufgrund dieser Arbeit verliehenen Titels berechtigt ist.

Bern, 13. Februar 2009

Gaudenz Steinlin