Universität Bielefeld
Fakultät für Soziologie
Sommersemester 2011

Arbeit zur Erlangung des Bachelor of Arts

Strukturen von Open-Source-Software-Projekten

Organisationsstrukturen im Medium Internet

Erstgutachten: Prof.'in Dr. Veronika Tacke

Zweitgutachten: Dr. Sven Kette

Eingereicht am: 27. Juli 2011

von

Jens Martin Heuer
Dölfesweg 8
30659 Hannover

jens_martin.heuer@uni-bielefeld.de

Inhalt

1. Einleitung

Folgt man dem Soziologen und Kulturtheoretiker Dirk Baecker, stellt der Computer bzw. das Internet ein neues Kommunikationsmedium dar, das in seinem Einfluss mit dem des Buchdrucks vergleichbar ist. Während der Buchdruck vor gut 500 Jahren die „Moderne" einläutete, markiert die Etablierung des neuen Mediums den Übergang zur „next society" (Drucker 2003). Wie jedes Kommunikationsmedium stellt es mehr kommunikative Möglichkeiten bereit, als durch die bestehenden Strukturen verarbeitet werden können und führt in der Folge zur Überforderung der Gesellschaft. „Die Gesellschaft […] bedarf daher so genannter Kulturformen, um das Mögliche auf das Bearbeitbare zu reduzieren" (Baecker 2007: 14).

Nicht zuletzt Organisationen sind von der Etablierung des neuen Verbreitungsmediums betroffen. In Anbetracht der neuen Kommunikationsmöglichkeiten, die das Internet bietet und die damit verbundene Steigerung der Komplexität und Dynamik von System-Umwelt-Beziehungen, erscheint ein Verhaften an etablierten Mitgliedschaftskonzepten und formalen Kommunikationswegen als Starrheit und fehlendes Reaktionsvermögen auf sich schnell verändernde Umweltgegebenheiten. Es stellt sich daher die Frage, welcher „Kulturformen" und Veränderungen der formalen Strukturen es in Organisationen bedarf, um auf die Erfordernisse des Mediums „Internet" reagieren zu können.

Open-Source-Software-Projekte stellen eine (Organisations-) Struktur dar, die eng mit dem Verbreitungsmedium Internet verbunden ist. In diesen Projekten schließen sich Softwareentwickler aus der ganzen Welt zusammen, um Software zu entwickeln, die meist kostenlos zur Verfügung gestellt wird und die auf dem Softwaremarkt mit Produkten etablierter Hersteller konkurriert. Ein Beispiel für solche Software ist der Webbrowser *Firefox*. 1998 stellte die Firma *Netscape* die Entwicklung an ihrem wenig erfolgreichen Webbrowser *Navigator* ein und überließ den Quellcode der freiwilligen Entwicklergemeinde, die sich in der *Mozilla Foundation* zusammenschloss und den weiter entwickelten Webbrowser unter dem Namen Firefox veröffentlichte. Heute ist Firefox der größte Konkurrent des Marktführers *Internet Explorer* aus dem Hause Microsoft.

Nun ist die Tatsache, dass eine Organisation mit ihren Produkten gegen Produkte einer anderen Organisation konkurriert –ihrer evolutionären Unwahr-

scheinlichkeit zum Trotz -- keineswegs überraschend für uns. Das Besondere an den Projekten ist jedoch, dass es sich augenscheinlich nicht um klassische Organisationen handelt. Die Projekte sind vielmehr lose gekoppelte Zusammenschlüsse freiwilliger Entwickler, die den Entwicklungszielen in unbezahlter Arbeit nachgehen. Die Projekte stellen offenbar nur geringe Anforderungen an die Mitgliedschaft und auch Hierarchien scheinen mehr oder weniger zu fehlen. Eine weitere Besonderheit der Projekte liegt darin, dass sie fast vollständig auf Face-to-Face Interaktionen verzichten und stattdessen in hohem Umfang die Möglichkeiten technisch vermittelter Kommunikation nutzen, die das Internet bietet.

Die Projekte, die bereits öfters Gegenstand soziologischer Untersuchungen waren, wurden auf Grund dieser vermeintlichen Inkompatibilität mit organisationalen Grundstrukturen zumeist unter einer netzwerktheoretischen Perspektive untersucht. Diese Perspektive verwehrt jedoch den Blick auf strukturelle Unterschiede zu klassischen Organisationen. Das Anliegen dieser Arbeit ist es daher, die Projekte aus einer organisationssoziologischen Perspektive zu untersuchen, spezifische Unterschiede zwischen herkömmlichen Organisationen und den Projekten herauszustellen und damit Anknüpfungspunkte für einen Vergleich der Strukturformen zu liefern. Zudem soll der Frage nachgegangen werden, welchen Einfluss das neue Kommunikationsmedium Internet auf die vorgefundenen Strukturen ausübt. Als empirisches Beispiel dient das Open-Source-Projekt KDE. Das KDE-Projekt erstellt eine graphische Oberfläche für Linux Betriebssysteme und zählt zu den größten und aktivsten Projekten im Bereich freier Software.

Die vorliegende Arbeit gliedert sich in vier inhaltliche Teile. Im ersten Abschnitt werden zwei wichtige gesellschaftsstrukturelle Voraussetzungen von Open Source Projekten dargestellt (3). Diese bestehen zum einen in der rechtlichen Verankerung von Open-Source-Softwarelizenzen, die das Urheberrecht dazu nutzen, die Software frei verfügbar zu machen und die Projekte erst dadurch ermöglichen (3.1). Anschließend wird das Massenmedium Internet und dessen Bedeutung für den Entwicklungsprozess beschrieben (3.2.). Danach folgt eine allgemeine Beschreibung des Projekts KDE, das als empirisches Beispiel dient, um im fünften Teil der Arbeit auf die Strukturen des Projekts einzugehen und

entlang von etablierten organisationssoziologischen Konzepten wie dem der Mitgliedschaft (5.1), Entscheidungsprämissen (5.2), Hierarchien (5.3) und Entscheidungen (5.4) zu untersuchen. Anschließend werden die Erkenntnisse zusammengefasst und in einen theoretischen Rahmen eingeordnet (6).

2. Methodenreflektion

Die Idee für die vorliegende Arbeit geht auf die Tätigkeit des Autors als Forschungsstudierender des Bielefelder „Instituts für Weltgesellschaft“ zurück. Die dortige Arbeit stand unter der Frage, ob es sich bei Open-Source-Softare-Projekten um eine globale Mikrostruktur im Sinne Knorr-Cetinas handele (Knorr-Cetina/Brügger 2002, sowie Knorr-Cetina 2005). Während die Beschreibung der Projekte als solcher Strukturtyp schwierig ist, gewann die Frage nach strukturellen Ähnlichkeiten und Differenzen zwischen den Projekten und organisierten Sozialsystemen immer größere Relevanz. Dieser Fragestellung solle in der hier vorliegenden Bachelor Arbeit nachgegangen werden.

Die Entscheidung, das Projekt KDE als exemplarisches Beispiel für Open-Source-Software-Projekte herauszugreifen erfolgte auf Grund dreier pragmatischer Kriterien. Zum einen ist KDE, gemessen an der Zahl der aktiven Mitglieder, eines der größten existierenden Open-Source-Software-Projekte. Während es bei kleinen Projekten passieren kann, dass die Mailinglisten übermehrere Wochen „verwaist“ sind, war bei einem Projekt dieser Größe und Entwicklungsdynamik klar, dass im Beobachtungszeitraum genügend auswertbares Material entstehen würde.
Der zweite Aspekt war, dass KDE bereits Gegenstand anderer Untersuchungen war (vgl. Taubert 2006/Brandt 2009). Somit stand über diese Quellen weiteres Material für eine sekundäre Auswertung zur Verfügung.
Zuletzt war das Projekt KDE thematischer Gegenstand meiner Tätigkeit am „Institut für Weltgesellschaft“, sodass ich die im Rahmen meiner dortigen Tätigkeit geführten Interviews ebenfalls für die veränderte Fragestellung nutzen konnte. Dabei stellte sich jedoch heraus, dass die damaligen Interviews die (Organisations-) Strukturen des Projekts nur wenig fokussierten. Lediglich ein Interview bat sich an, für die neue Fragestellung herangezogen zu werden.

Die Analyse der Strukturen erfolgte primär durch die Beobachtung und Analyse der gängigen Kommunikationsmedien des Projekts. Die Mailinglisten und Blogs stellen eine sehr interessante Quelle dar, da diese öffentlich zugänglich sind und man die projektinterne Kommunikation so ungefiltert beobachten kann. Man ist somit nicht auf (Selbst-) Beschreibungen durch Mitglieder in Form von Interviews angewiesen.

Die Beobachtung der Mailinglisten und Blogs erfolgte zunächst ohne Einschränkung auf spezifische Themen o.ä. Schnell wurde jedoch klar, dass auf Grund des hohen und nahezu unüberblickbaren Kommunikationsvolumens nur eine selektive Beobachtung praktikabel war.

Die weiteren Beobachtungen der Mailinglistenkommunikation wurden daher auf die Mailingliste „core-devel“, auf der grundsätzliche Entscheidungen, wie die Festlegung von Veröffentlichungsterminen diskutiert werden, sowie die Mailingliste des Subprojekts KDE-Pim, eingeschränkt.[1]

Um die Mailinglistenkommunikation für die hier verfolgten Zwecke zu nutzen, musste von den spezifischen Themen, die in den Diskussionen verfolgt wurden, abstrahiert werden. Es ging nicht darum, technische Argumentationen in der Entwicklung nachzuvollziehen. Stattdessen wurden die Diskussionen genutzt um explorativ Strukturen des Projekts zu identifizieren.

Dabei wurden Fragestellungen, wie, ob und wenn ja, auf welche Weise (formale) Erwartungen in den Diskussionen aktualisiert werden; wie Entwicklungen zugerechnet werden; auf welche Weise Entscheidungen im Projekt getroffen werden; ob es zur Bildung bestimmter Interessensgruppierungen kommt; in welcher Weise Träger besonderer Rollen, wie die Maintainer, in Entscheidungsprozesse eingreifen oder ob eine Standardisierung der Entscheidungsprozesse beobachtbar ist, relevant.

[1] Die Mailinglisten sind unter lists.kde.org einsehbar.

3. Allgemeine Besonderheiten von Open-Source-Software-Projekten

Das folgende Kapitel soll einige allgemeine Besonderheiten des Felds der Open-Source-Software aufzeigen und die enge Bindung der Projekte an Recht und Massenmedien illustrieren. Besonderes Augenmerk wird auf die lizenzrechtlichen Besonderheiten von Open-Source-Lizenzen, dem sog. Copy*left,* gelegt (3.1). Die „vertragliche Innovation freier Softwarelizenzen“ (Taubert 2006: 65ff.) besteht darin, dass sie das Entwicklungsmodell freier Software durch die spezielle Reglung des Urheberrechts erst ermöglichen. Die unter diesen Lizenzen veröffentlichte Software wird zu einem „Kollektivgut“ (Taubert 2006: 116), wodurch die soziale Exklusion aus dem Entwicklungsprozess praktisch ausgeschlossen wird. Die in der Bewegung geteilten Normen in Bezug auf freie Verfügbarkeit von Wissen werden auf diese Weise in eine vertragliche Form gebracht und erhalten rechtliche Wirksamkeit.
Erst durch die Verwendung der technischen und kommunikativen Möglichkeiten des Mediums Internet kann die vertraglich zugesicherte Inklusion jedoch auch realisiert werden (3.2). Mit Hilfe von one-to-many Medien, wie Mailinglisten, Blogs und Chats können die Schranken kommunikativer Erreichbarkeit gesenkt werden. Gleichzeitig konstituieren diese Medien auf Grund ihrer Schriftlichkeit das Gedächtnis der Projekte.

3.1 Rechtliche Fundierung

Beschäftigt man sich mit Open Source Software, so kann man dies nicht tun, ohne einen Blick auf das damit verbundene Lizensierungsmodell zu werfen. Während die Rechtsdogmatik das herkömmliche Urheberrecht „als Mittel des Ausgleichs zwischen den Interessen unterschiedlicher Parteien betrachtet“ (Taubert 2006: 78) und so als Anreiz schöpferischer Tätigkeit verstanden wird, wird es im Kontext freier Software „nicht als Instrument der Förderung schöpferischer Tätigkeit, sondern im Gegenteil als störendes Hindernis wahrgenommen“ (ebd.: 72).

Die Beschreibung der restriktiven Urheberrechte als dysfunktional ist auf den Bruch, den die beginnende Kommerzialisierung von Software am Anfang der 1980er Jahre auslöste zurückzuführen.

> „Als ich 1971 anfing, im Labor für Künstliche Intelligenz des MIT zu arbeiten, wurde ich Teil einer gemeinschaftlich Software nutzenden Gesellschaft, die seit vielen Jahren existiert hatte. Das miteinander Teilen von Software war nicht auf unsere einzelne Gruppe beschränkt; es ist so alt, wie die Computer selbst [...] Wann immer Leute einer anderen Universität oder eines Unternehmens ein Programm portieren und nutzen wollten, erlaubten wir es ihnen gern. Wenn man jemanden ein ungewohntes und interessantes Programmnutzen sah, konnte man immer fragen, ob man den Quellcode bekam, man konnte ihn lesen, ändern, ihn ausschlachten und Teile davon nutzen um, ein neues Programm zu schreiben.“ (Stallman 1998: o.S.)

Diese Situation änderte sich zu Beginn der 80er Jahre. Zum einen waren kommerzielle Softwareanbieter, wie Microsoft, Novell oder Lotus in den Softwaremarkt eingestiegen und vertrieben ihre Programme als Binär-Versionen, in denen der Quellcode nicht einsehbar und dementsprechend nicht veränderlich war. Zudem untersagten die Lizenzen auch die Vervielfältigung.
„Das bedeutete, dass der erste Schritt beim Nutzen eines Computers darin bestand, zu versprechen, seinem Nachbarn nicht zu helfen. Eine kooperierende Gemeinschaft war verboten“ (ebd.).

Die durch Urheberrechte verursachten, dysfunktionalen Folgen können in drei Bereichen auftreten. Erstens wird der Prozess der Softwareentwicklung selbst durch Urheberrechte negativ beeinflusst, zweitens ist die Nutzung der Software Restriktionen unterworfen und drittens hat es Auswirkungen auf die sozialen Beziehungen zwischen den Entwicklern (vgl. Taubert 2006: 72 ff.).

Die Entwicklung von Software war ein kooperativer und evolutionärer Prozess, in dem die Entwicklung von Software dadurch geprägt war, dass unterschiedliche Entwickler sukzessiv neue Funktionen in ein bestehendes Programm implementierten oder Programmbestandteile Ausgangsbasis für neue Programme bildeten. Dieses Modell ist durch das Auftreten von Urheberrechten und dem damit verbundenen Verbot Software zu modifizieren, unmöglich geworden (vgl. Stallman 1992).
Eric Raymond, einer der „philosophischen“ Begründer der Open-Source-

„Bewegung“[2] beschreibt die entstehende Problematik folgendermaßen: „Creative brains are a valuable, limited resource. They shouldn’t be wasted on re-inventing the wheel when there are so many fascinating new problems waiting out there.” (Raymond 2001: 3.2.2). Genau dies wird jedoch aus Sicht der Initiatoren des Open-Source-Gedankens, durch proprietäre Software bewirkt. Es ist nun notwendig Probleme wiederholt zu lösen, da auf bereits etablierte Lösungen mangels Einblick in den Quellcode nicht zurückgegriffen werden kann. Folgt man der Argumentation Raymonds, gestaltet sich Softwareentwicklung unter Bedingungen restriktiver Softwarelizenzen ausgesprochen ineffizient.

Der zweite Aspekt bezieht sich darauf, dass es als einer der größten Vorteile von Software gesehen wird, dass diese von Nutzern entsprechend ihrer Bedürfnisse angepasst werden kann. Stallman (1992) zufolge geht dieser Vorteil bei kommerzieller Software jedoch verloren. Ohne Einblick in den Quellcode bleibt die Software für ihren Nutzer eine „black box“ (ebd.). Dieser hat in der Folge keine Kontrolle mehr über das Programm, sodass sich ein Abhängigkeitsverhältnis zum Hersteller etabliert.

Zuletzt haben urheberrechtlich geschützte, kommerzielle Softwarelizenzen auch einen Einfluss auf die Gemeinschaft der Entwickler. Stallman (1985) fasst die Konsequenzen folgendermaßen zusammen:

> “Many programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends. The purchaser of software must choose between friendship and obeying the law.”

Die „Unterbindung des gruppenkonstituierenden Austauschakts“ (Taubert 2006: 75) durch restriktive Lizenzen führt zu konfligierenden Verhaltenserwartungen zwischen Gesetz und Peer-Group und „erweist sich so als bedrohlich für die soziale Kohäsion“ (ebd.).

[2] Zur Frage, ob die Open-Source-Community eine soziale Bewegung darstellt siehe Zimmermann 2004.

Um dem entgegenzuwirken nutzen die freien Softwarelizenzen das Urheberrecht und sichern die Freiheit der Software vertraglich. Die bekannteste und verbreitetste Open Source Lizenz ist die GNU GPL. Diese räumt weitgehende Nutzungsrechte ein, verhindert jedoch, „der Software eigene Begrenzungen hinzuzufügen“ (Stallman 1998 o.S.). Die Software darf in vollem Umfang genutzt, vervielfältigt, verbreitet und modifiziert werden. Die modifizierte Programmversion steht durch eine sog. Repititionsklausel unter der Anforderung, der Gemeinschaft wieder zugänglich gemacht und ebenfalls unter dem „Copyleft“ (ebd.) der GNU GPL lizensiert zu werden. Auf diese Weise soll ausgeschlossen werden, dass Open Source Software nach ihrer Weiterentwicklung mit einem Copyright belegt wird.
„Copyleft nutzt Copyright-Gesetze, aber dreht sie um, um auf gegenteilige Weise zu wirken, als deren gewöhnliche Anwendung: anstatt als ein Mittel um Software zu privatisieren, wird es ein Mittel um Software frei zu halten.“ (ebd.) „Die Pointe“, so Holtgrewe (2005: 5), „ist ihr ansteckender Charakter: Die Lizenz legt fest, dass der Quellcode beliebig weitergegeben und verändert werden darf, aber dass der veränderte Quellcode bei der Weitergabe wieder unter derselben Lizenz veröffentlicht werden muss.“ Die so lizensierte Software wird in der Folge „faktisch zum Kollektivgut“(Taubert 2006: 116) und jede Veränderung des Quellcodes zur „Gabe“ (ebd.) an die Gemeinschaft.

Damit führt die GPL eine wichtige Neuerung gegenüber älteren Open-Source-Lizenzen, wie beispielsweise der BSD-License, ein. Bei dieser Lizenz war es möglich, den Code nach seiner Veränderung unter eine restriktive Lizenz zu stellen. Dies wurde z.B. von Apple praktiziert, als das Unternehmen in der Mitte der neunziger Jahre ein unter der BSD-Lizenz stehendes UNIX-System nach diversen Veränderungen unter eine restriktive Lizenz stellte und als Mac OSX veröffentlichte.

Eine kommerzielle Nutzung der unter der GPL lizensierten Software ist jedoch nicht ausgeschlossen. Freie Software wird in der Szene „im Sinne von ‚free speach not free beer‘“ verstanden (Stallman zitiert nach Holtgrewe 2005: 1). Das entscheidende Kriterium ist demnach die Offenheit des Quellcodes, sowie die Freiheit, die Software modifizieren, nutzen und verbreiten zu können. Entscheidend ist hingegen nicht, dass die entsprechende Software kostenlos zur

Verfügung gestellt wird, auch wenn dies in den meisten Projekten der Fall ist. Richard Stallman finanzierte das GNU Projekt Anfangs durch den Verkauf von Bandkassetten mit GNU-Software. Eine kommerzielle Nutzung in dieser Form ist heute nur noch eingeschränkt möglich, da die hohe Verbreitung von Breitband-Internetanschlüssen den Bezug über kostenfreie Online-Angebote attraktiver macht. Es besteht aber die Möglichkeit Open Source Software indirekt zu kommerziellen Zwecken zu nutzen, z.B. indem Support-Leistungen verkauft werden, wie es die Firmen Novell oder Red Hat praktizieren.

Auch von Firmen kann die Lizenz zur Distribution ihrer Software verwendet werden. So entwickelt der Konzern Google sein Betriebssystem für Mobiltelefone Android unter einer Open Source-Lizenz. Das System, das auf Linux aufbaut, wurde so innerhalb kürzester Zeit auf verschiedene weitere Architekturen portiert und zahlreiche Funktionen durch freiwillige Mitarbeiter implementiert. Die Open Source-Lizensierung ermöglicht eine schnelle und User orientierte Entwicklung der Plattform, da Nutzer gleichzeitig zu Entwicklern werden und gewünschte Funktionen implementieren können.[3] Außerdem fallen für Hardwarehersteller – und damit auch für die Verbraucher – Lizensierungskosten weg, wodurch die Geräte, im Vergleich zu denen von Apple und Microsoft, günstiger auf den Markt gebracht werden können. Wie Zahlen des US-amerikanischen Meinungsforschungsinstituts *NPD* belegen, ist diese Strategie sehr erfolgreich. Android-Geräte sind seit dem ersten Quartal 2010 - knapp ein Jahr nach Markteinführung - mit einem Marktanteil von 28% weiter verbreitet, als die der Konkurrenten Apple und Microsoft (vgl. pcgames.de 2010). Lediglich der Blackberry-Hersteller RIM weist derzeit (noch) einen höheren Marktanteil auf.
Durch die Nutzung der Open-Source-Software schafft sich Google eine engagierte und scheinbar auch loyale Nutzerbasis. Google macht jedoch mit Android keinen direkten Profit. Software stellt – im Gegensatz zu Apple oder Microsoft – aber auch nicht Googles Haupteinnahmequelle dar. Die Gewinne erwirtschaftet der Konzern durch userspezifische Werbeplatzierungen, für die im Mobilfunksegment weitere Daten gesammelt werden können. Google kann seinen Wer-

[3] Dieser Aspekt wird ausführlicher unter 5.1 thematisiert

bekunden so weitere Werbemöglichkeiten und Daten, die ein spezifischeres „Targeting“ ermöglichen, offerieren. Open-Source-Software findet auch hier nur indirekt kommerzielle Verwendung.

3.2 Virtuelle Kommunikation

Während Open Source-Lizenzen, wie die GNU GPL, dafür sorgen, dass prinzipiell jeder an der Entwicklung der Projekte teilhaben *darf*, ist es auf den Einsatz von internetbasierten Kommunikationstechnologien zurückzuführen, dass prinzipiell auch jeder an den Projekten mitarbeiten *kann* – vorausgesetzt es besteht eine Internetverbindung mit ausreichenden Kapazitäten. Diese Einschränkung kann jedoch nicht auf interne Strukturen zurückgeführt werden.
Open-Source-Projekte greifen in hohem Maß auf die technische Infrastruktur des Internets zurück, wodurch auf Face-to-Face Interaktion weitgehend verzichtet werden kann. Stattdessen wird schriftliche Kommunikation über Mailinglisten, Foren, Blogs und Chats genutzt. Das hat den strukturellen Vorteil, dass kommunikative Beteiligung nicht auf körperliche Anwesenheit angewiesen ist. Ohne die Möglichkeiten zur dezentralen und asynchronen Kommunikation, die das Internet bietet, wäre das Entwicklungsmodell von Open Source Projekten nicht möglich.

Bereits die Announcements der Projekte GNU und Linux fanden in sog. *Newsgroups* statt. Diese stehen, wie auch die heute verwendeten Systeme, wie Mailinglisten, IRC-Chats oder Blogs, als one-to-many-Medien für ein hohes Maß sozialer Inklusion.
Im Gegensatz zu Interaktionen oder Organisationen, welche immer auch einen räumliche Bezug in der realen Welt haben (Produktionsanlagen, Büroräume, das Treffen in der Kneipe etc.) fehlt ein solcher Bezug in Open-Source-Projekten vollständig. Der Zugang zum Interaktionsraum der Projekte ist nicht von einem physischen Zugang abhängig, sondern setzt lediglich entsprechende technische Ausrüstung – Internetanschluss und Computer – voraus. So wird es möglich, räumlich und auch zeitlich weit voneinander entfernte Akteure in den Entwicklungsprozess einzubeziehen. Taubert (2006: 45) spricht daher von einer „Enträumlichung des Entwicklungsprozesses“, bei der „‚translokale‘ Orte, an

denen Gleichgesinnte mit ähnlichen Interessen und Kompetenzen in einer Weise korrespondieren, als säßen sie sich gegenüber.“ (Hoffmann 1998: o.S.) entstehen.

Welche Medien genutzt werden variiert von Projekt zu Projekt. Das meist genutzte und vermutlich in allen Projekten vorzufindende Medium ist die *Mailingliste*. Mailinglisten sind im Prinzip E-Mail-Verteiler und in der Regel öffentlich zugänglich. Man kann sich über ein Formular auf der Website des entsprechenden Projekts in den Verteiler der Listen eintragen und die Projektkommunikation direkt und ungefiltert mitverfolgen. Zudem ist auch eine aktive Beteiligung an den Diskussionen möglich. Technisch bestehen keine Restriktionen, sodass Redebeiträge auch von nicht-Entwicklern stammen können.[4] Von den kommunikativ in Erscheinung tretenden Akteuren ist nur die E-Mail-Adresse sichtbar, welche maximal Rückschlüsse auf die Merkmale Geschlecht und mit einschränkungen Herkunft ermöglicht. Auf weitere Merkmale wie Alter, Bildungshintergrund, Vermögen oder soziale Beliebtheit kann hingegen nicht geschlossen werden. Jim Blandy (zitiert nach Fogel 2007:78), ein langjähriger Open-Source-Entwickler, illustriert diese Unmöglichkeit das „Gegenüber“ sozialstrukturell zu kontextualisieren:

> “Back in 1993, I was working for the Free Software Foundation, and we were beta-testing version 19 of GNU Emacs. We’d make a beta release every week or so, and people would try it out and send us bug reports. There was this one guy whom none of us had met in person but who did great work: his bug reports were always clear and led us straight to the problem, and when he provided a fix himself, it was almost ways right. He was top-notch.
> Now, before the FSF can use code by someone else, we have them do some legal paperwork to assign their copyright interest to that code to the FSF. Just taking the code from complete strangers and dropping it in is a recipe for legal disaster.
> So I emailed the guy the forms, saying, ‘Here’s some paperwork we need, here’s what it means, you sign this one, have your employer sign that one, and we can start putting in your fixes. Thanks very much.’
> He sent me back a message saying, ‘I don’t have an employer.’
> So I said, ‘Okay, that’s fine, just have your university sign it and send it

[4] Dies kann jedoch auch dysfunktionale Folgen haben. Fogel (2007: 89f.) berichtet von einem Projekt, in dem die drittmeisten Beiträge auf der Mailingliste stammten. So wurde in dem Projekt viel Aufmerksamkeit in nicht entwicklungsrelevanten Diskussionen gebunden und die Entwicklung blockiert.

> back.'
> After a bit, he wrote me back again, and said, 'Well, actually… I'm thirteen years old and I live with my parents.'"

Äußere Merkmale der Akteure spielen auf Grund ihrer nicht-Bekanntheit keine Rolle im Entwicklungsprozess. Mit anderen Worten: auf der Grundlage äußerer Merkmale kann nicht zwischen den Entwicklern diskriminiert werden. Die Diskussionen stützen sich einzig auf technische Argumente, welche im Verlauf der Diskussion hervorgebracht werden und sehen von anderen Merkmalen ab. Die Identitäten der beteiligten Akteure konstituieren sich daher lediglich durch ihre jeweiligen Diskussions- und Codebeiträge. Dies schließt jedoch nicht aus, dass die beteiligten Akteure Annahmen übereinander bilden, wie im Fall des 13-jährigen Jungen, bei dem eine berufliche Anstellung oder ein universitäres Studium unterstellt wurde, vermutlich, weil dies dem Lebenskontext der meisten Entwickler entspricht.

Alle Medien, die zur Kommunikation in den Projekten verwendet werden, sind Massenmedien. Unter dem Begriff der Massenmedien sind nach Luhmann (2009:10) „alle Einrichtungen der Gesellschaft [zu fassen], die sich zur Verbreitung von Kommunikation technischer Mittel der Vervielfältigung bedienen", d.h., dass sie one-to-many-Medien sind.
Der Kreis der Empfänger kann in diesen Medien nicht vorherbestimmt werden. Stattdessen richtet sich jeder Kommunikationsakt an die ganze Welt. Welt wird hierbei im Sinne Luhmanns, als „Letzthorizont allen Sinns" (Luhmann 1987: 105), als Einheit der durch einen Beobachter eingeführten Differenz zwischen System und Umwelt verstanden (vgl. Esposito et.al. 1997: 205). Die Gruppe der Empfänger kann nicht eingegrenzt werden, stattdessen transzendieren die verwendeten Kommunikationsmedien jede, in Bezug auf die kommunikative Erreichbarkeit getroffene Differenz.
Auch wenn ein Beobachter jede Kommunikation als Kommunikation durch Mitglieder beobachtet, weißt sie doch eine viel höhere Reichweite auf. Jeder kann, sofern ein Internetzugang zur Verfügung steht, an diesen öffentlich zugänglichen Kommunikationen partizipieren – nicht nur als Entwickler.

Dies bedeutet für die Kommunikation in Open-Source-Software-Projekten, dass, „eindeutige Präsenz durch Unterstellung ersetzt werden muss" (Luhmann

2009: 12) und auf den unwahrscheinlichen Erfolg zweier Selektionen angewiesen ist: „die Sendebereitschaft und das Einschaltinteresse, die nicht zentral koordiniert werden können." (ebd.: 11)

Sofern die Einschätzung zutreffend ist, dass Open-Source-Software-Projekte für ihre Kommunikation massenmediale Strukturen nutzen, so ist anzunehmen, dass sich in den Strukturen der Projekte die den Massenmedien inhärente Präferenz für Information - und damit Neuheit - wieder findet. Denn: „Eine Nachricht, die ein zweites Mal gebracht wird, behält zwar ihren Sinn, verliert aber ihren Informationswert." (Luhmann 2009: 31)

Im Gegensatz zu klassischen Software-Herstellern, bei denen ein Major-Release einen „Life-Cycle" von mehreren Jahren aufweist, sind Open-Source-Projekte von einem dauerhaften Wandlungsprozess und ständiger Innovation und Neuheit geprägt. Diese kann die Form von erweiterter Funktionalität, der Verbesserung bestehender Programmbestandteile oder der Initiierung neuer Programme annehmen. Neuheit meint dabei auch relative Neuheit, also imitative Innovation. Der Prozess des ständigen Wandels ist nicht zuletzt, so die hier verfolgte These, durch die verwendeten Kommunikationskanäle bedingt, in denen Neuheit Aufmerksamkeit zu binden vermag und mittels derer dauerhaft neue Umweltreize in das Projekt hineingelangen können.

4. Das Projekt KDE

Das Projekt KDE (ursprünglich: Kool-Desktop-Environment) wurde 1996 von dem deutschen Informatikstudenten Mathias Ettrich initiiert, um eine am Endnutzer orientierte, umfassende graphische Benutzeroberfläche für Linux zur Verfügung zu stellen.[5] Dies war zu dem Zeitpunkt der Projektgründung ein innovatives Anliegen, da Vergleichbares unter Linux nicht zur Verfügung stand. Heute zählt KDE, die Zahl der aktiven Entwickler (ca. 2400[6]) und die der Benut-

[5] Siehe hierzu die Projektankündigung im Usenet durch Mathias Ettrich (1996), der 2009 für seine Verdienste im Bereich der Open-Source-Software mit dem Bundesverdienstkreuz ausgezeichnet wurde (Vgl. Landespressestelle Berlin 2009)
[6] Vgl. Albers 2010

zer betreffend, zu den größten Projekten im Bereich der Open-Source-Software.

Die stark wachsende Entwicklergemeinde hatte und hat einen Zuwachs an Funktionen und Applikationen zur Folge. So beschränkt sich KDE inzwischen nicht mehr auf die Herstellung einer graphischen Oberfläche, sondern stellt zudem viele Programme, die zur produktiven Nutzung eines Computers notwendig sind, zur Verfügung. Zum Software-Umfang von KDE gehören heute unter anderem ein Webbrowser (Konqueror), ein PDF-Betrachter (Okular), eine Offce-Suite (KOffce), eine Programmierumgebung (KDevelop), diverse Spiele und viele weitere Programme. Außerdem ist es möglich KDE als alternative Desktop-Oberfläche unter Windows und Mac OSX zu betreiben. Die gesamte KDE-Software ist unter der GPL lizensiert.

Die zunehmende Differenzierung, Ausweitung und Professionalisierung der Software hatte eine Veränderung der Marketingstrategie zur Folge, die seit 2009 verfolgt wird (vgl. Jarvis 2009). Seither wird mit dem Begriff KDE nicht mehr das Produkt, sondern die entwickelnde Community bezeichnet. Die Software selbst wird im Rahmen der KDE-SC (Software-Compilation) ausgeliefert.

Der steigende Funktionsumfang wurde von einer starken internen Differenzierung in viele Subprojekte begleitet. Insgesamt siedeln sich um das Projekt inzwischen 24 Unterprojekte an. Die Differenzierung kann in drei Kategorien unterteilt werden:
Die erste Gruppe sind die Programme der *KDE-Workspaces*. „These provide the environment for running and managing applications and integrate interaction of applications." (ebd.: o.S.) In den Subprojekten dieser Kategorie wird dem ursprünglichen Zweck des Projekts nachgegangen und eine Benutzeroberfläche in verschiedenen, angepassten Varianten (z.B. für Desktop Computer oder Netbooks) entwickelt. Neben der Differenzierung in verschiedene, angepasste Oberflächen finden sich hier außerdem grundlegende Systemprogramme, wie die Systemeinstellungen, der Fenstermanger KWin und ähnliches.
Die Kategorie *KDE-Platform* umfasst Programmbibliotheken und -services, die grundlegende Funktionen bereitstellen, die von jeder KDE Software genutzt werden. Hierzu gehören das Multimedia-Framework Phonon, der Datenverwal-

tungsdienst Akonadi, die Plasma-Bibliothek und einige weitere.
Die Differenzierung der dritten Gruppe *KDE-Applications* verläuft entlang von Programmen bzw. Programmgruppen. So sammelt sich z.B. im Subprojekt *KDE-Edu* Lernsoftware wie der Globus Marble oder die Sternenkarte KStars. Das Subprojekt Amarok entwickelt hingegen ein Programm zur Verwaltung und Wiedergabe von Musikdateien – vergleichbar mit dem Funktionsumfang von iTunes. Hinzu kommen viele weitere Programme.

Im Projekt findet „Systembildung im System“, also der Aufbau höherer Komplexität durch Reduktion von Komplexität statt. Während die zunehmende Differenzierung für das Gesamtprojekt eine Steigerung des Funktionsumfangs und damit gestiegener Komplexität bedeutet, da sowohl der Umfang des Programmcodes gestiegen ist, wie auch die Abhängigkeiten zwischen einzelnen Softwarebestandteilen, bietet die Differenzierung für die Subprojekte eine massive Entlastung. Ihnen ist es möglich auf die bereits etablierte Infrastruktur, Programmbibliotheken und –services des Gesamtprojekts zuzugreifen. Unter der Infrastruktur werden hier die vom Gesamtprojekt zur Verfügung gestellten Serverkapazitäten, Kommunikationsstrukturen, wie Mailinglisten und Distributionsmöglichkeiten durch die Integration in die Software-Compilation verstanden.
Für ein Projekt, das sich selbst als ein Subprojekt von KDE initiiert, statt die Programmentwicklung „auf eigene Faust“ zu versuchen, bedeutet die Eingliederung in ein Hauptprojekt massive Reduktion von Komplexität. Komplexe Entscheidungssituationen, wie die, welche Programmiersprache verwendet, unter welcher Lizenz die Software veröffentlicht oder wie der Quellcode bereitgestellt werden soll, stellen sich für diese Projekte nicht. Zudem können die (Sub-) Projekte auf Entwicklungen anderer (Sub-) Projekte zurückgreifen und werden dadurch von vielen Entwicklungsschritten entlastet, die andernfalls vollzogen werden müssten. So kann das Design der Programm Icons den Graphikern und Entwicklern des Subprojekts *KDE-Artwork* überlassen werden, die Entwicklung der Programmbibliotheken findet in den *KDE-Platform*-Projekten statt usw. Diese Struktur ermöglicht es Kapazitäten insofern optimal zu nutzen, als dass Entwickler sich in den Subprojekten auf ihre „Kernkompetenz“ konzentrieren können. Dies bedeutet jedoch nicht, dass ein Entwickler des KDE-Multimedia-Projekts kein Mitspracherecht bei Artwork Angelegenheiten hat. Sofern der

Entwickler sich die zeitlichen Ressourcen nimmt, um an den Entscheidungsprozessen zu partizipieren, kann er sich auch hier engagieren. Mit anderen Worten: das Engagement in einem Subprojekt ist nicht exklusiv.
Neben der Entlastung von komplexen Entscheidungssituationen führt eine solche Eingliederung in ein großes Projekt außerdem dazu, dass ein noch junges Projekt von dem Prestige des Gesamtprojekts profitieren kann und auf diese Weise schneller zu (relativ) hohen Nutzerzahlen kommen kann. Hohe Nutzerzahlen machen das Projekt in der Folge attraktiver für freiwillige Entwickler, sodass die Entwicklung in der Regel schneller vorangetrieben werden kann.

Die vom Projekt KDE entwickelte Software folgt seit dem Versionssprung auf 4.0 einem halbjährlichen Veröffentlichungs-Zyklus (vgl. Albers 2010). Zwischen Ende Januar und Anfang Februar bzw. Ende Juli und Anfang August jeden Jahres wird ein neues „Major-Release“ veröffentlicht, das die zu diesem Zeitpunkt stabile Software aller Subprojekte umfasst. In der Zwischenzeit werden monatliche „Bugfix-Releases“ veröffentlicht, in denen zwar die von Nutzern gemeldeten Fehler entfernt, jedoch keine neuen Funktionen hinzugefügt werden. Zeitgleich zur Vorbereitung eines Major-Releases beginnt bereits die Entwicklung der nächsten Version.

Neben dem Projekt KDE existiert der KDE e.V., der als juristische Person die Finanzen des Projekts verwaltet und die technische Infrastruktur (z.B. Server) finanziert und bereitstellt und die Rechte an der Marke KDE hält. Der Verein nimmt auf die Entwicklungstätigkeit im Projekt jedoch keinen Einfluss und wird daher im Folgenden nicht weiter thematisiert.

5. Die Strukturen des Projekts KDE

5.1 Mitgliedschaft im Projekt KDE – Der Entwicklerstatus

Anders als in Interaktionssystemen, in denen die bloße Anwesenheit genügt um Teil des Systems zu sein, ist die Inklusion in ein organisiertes Sozialsystem an die Rolle des Mitglieds geknüpft. In dieser kondensiert ein Set an Verhaltenserwartungen, die festlegen, was von einem Mitglied erwartbar, aber auch, was

nicht-erwartbar ist. Zu diesen Erwartungen gehört die Unterwerfung unter die Autorität, die Übernahme von im Voraus nicht spezifizierten Aufgaben uvm. Als Gegenleistung erhält das Mitglied Geldzahlungen von der Organisation. Organisationen können so ein System „bezahlter Indifferenz“ (Luhmann 1999: 96) etablieren.

Da die Entscheidung Mitglied einer Organisation zu werden auf Freiwilligkeit beruht, steht dem Mitglied immer auch die Möglichkeit offen, die Mitgliedschaft zu beenden, sofern Nicht-Mitgliedschaft attraktiver erscheint als Mitgliedschaft. Solange diese Option nicht gewählt wird, kann von Seiten des Systems unterstellt werden, dass Mitgliedschaft attraktiver erscheint als Nicht-Mitgliedschaft und somit die generelle Motivation besteht die Mitgliedsrolle aufrecht zu erhalten.

Durch die Absonderung der „Motivation der Mitglieder von anderen Systemproblemen [...], ergeben sich im Weiteren äußerst bedeutsame Effekte für die Organisation. Zunächst wird die der Organisation zuzurechnende Kommunikation von der Funktion entbunden Mitglieder motivieren zu müssen“ (ebd.: 12). Organisationen können ihren Mitgliedern daher ein gewisses Maß an „Veränderungen, Enttäuschungen und Belastungen“ (ebd.: 95) aufbürden, bevor sich Mitglieder dazu entschließen ihren Mitgliedsstatus aufzugeben. Dies ermöglicht Organisationen sich an verändernde Umweltbegebenheiten anzupassen und in der Folge die Verhaltenserwartungen an ihre Mitglieder zu ändern. Erkauft wird diese Möglichkeit mit der resultierenden Folgeproblematik einer getrennten Teilnahme- und Leistungsmotivation (vgl. ebd.: 104 ff.).

Der Organisation wird durch die Kopplung von Verhaltenserwartungen an formalisierte Ein- und Austrittsbedingungen ermöglicht, „hochgradig künstliche Verhaltensweisen relativ dauerhaft zu reproduzieren“ (Luhmann 1975). In der Folge werden auch unwahrscheinliche Verhaltensweisen mit Wahrscheinlichkeit ausgestattet, was den Aufbau und Erhalt komplexer Strukturen ermöglicht. Durch die Formulierung von Verhaltenserwartungen an Mitglieder konstituiert ein organisiertes Sozialsystem seine Grenzen.

Stellt man die Mitgliedschaftsfrage für das Projekt KDE, so bieten sich zwei alternative Erklärungsansätze. Folgt man der Selbstbeschreibung von KDE als

Community, so schlösse diese Mitgliedschaftsdefinition auch diejenigen mit ein, die die Software nur nutzen und sich sporadisch an Diskussionen rund um KDE betreffende Themen beteiligen. Ein solch weiter Mitgliedschaftsbegriff macht insofern Sinn, als dass im Projekt präsente Verhaltenserwartungen auch für diese charakterisierte Gruppe gültig sind, sofern sie über die vom KDE-Projekt bereitgestellten Kanäle (Mailinglisten, Foren, IRC-Chats) kommunizieren. Gleichzeitig erweist er sich jedoch als schwierig, da Mitgliedschaft zu einer amorphen, schwer greifbaren Masse wird.

Daher wollen wir hier auf einen alternativen Begriff der Mitgliedschaft abstellen. Dieser macht sich eine technische Einrichtung des KDE-Projekts zu nutze. Die Entwicklung erfolgt über eine sog. Versionsverwaltung. Das Projekt KDE nutzt die Versionsverwaltungssoftware Subversion, kurz SVN. Während der Quellcode hier öffentlich zur Verfügung steht, gelesen und heruntergeladen werden kann, ist der Schreibzugriff auf den Server limitiert. Dieser ist nur denjenigen möglich, die über ein Benutzerkonto verfügen, über das sie identifiziert werden können.[7] Man kann, wenn man den Fokus auf den Entwicklungsprozess richtet, sagen, dass (aktives) Mitglied des Projekts ist, wer über einen Account verfügt, also Schreibzugriff auf den Quellcode hat. Diese Definition der Mitgliedschaft im Projekt deckt sich mit der vom Projekt verwendeten Definition der Rolle des *Developers.*

Möchte man sich als aktiver Entwickler am Projekt beteiligen, ist man zunächst gezwungen seine lokal (d.h. am eigenen PC) erstellten Änderungen („Patches“) der Software an den „Maintainer“[8], die Mailingliste oder das „Review Board“[9] des entsprechenden Subprojekts zu senden. Dort werden die Änderungen von Erfahrenen Entwicklern geprüft und, sofern sie den technischen Erfordernissen entsprechen und nicht zu Instabilitäten im bereits vorhandenen Code führen, in den Quellcode implementiert. Diese Möglichkeit der aktiven Beteiligung am Projekt steht *jedem* offen. Interessenten steht auf den Internetseiten des KDE-Projekts eine umfassende Dokumentation der Programmiersprache, sowie di-

[7] Hier muss eine gültige Email-Adresse hinterlegt werden. Vgl.: KDE.org 2010b
[8] Maintainer sind Entwickler, die die (Sub-) Projekte betreuen. Siehe hierzu auch Kap.3.3
[9] Das „Review Board“ ist eine eigens zum Peer-Review von Patches eingerichtete Website. Einsehbar unter: https://git.reviewboard.kde.org/r/

verse Tutorials zur Verfügung, die die Hürden für den Einstieg senken sollen. Zudem können die Kommunikationskanäle des Projekts genutzt werden, um sich bei erfahreneren Entwicklern Hilfestellungen bei der Lösung komplexer Probleme zu holen.

Engagiert sich jemand über einen längeren Zeitraum im Projekt und die Codebeiträge weisen eine gewisse Qualität auf, kann ein eigener SVN-Account erstellt werden, der den Schreibzugriff auf die Server ermöglicht. Patches können nun direkt in den Quellcode implementiert werden. Zur qualitativen Kontrolle des übermittelten Codes hat sich in dem Projekt ein Peer-Review Verfahren etabliert. Dabei wird der Quellcode ständig von allen Entwicklern auf mögliche Fehler und Inkonsistenzen geprüft und Modifikationen implementiert.

Die Entscheidung Entwickler im Projekt zu werden, ist zunächst einseitig. Will man sich am Projekt beteiligen, kann man dies in genannter Weise tun. Mitgliedschaft wird über Selbstrekrutierung erlangt, d.h. die Entscheidung wird vom Mitglied, nicht aber von dem Projekt getroffen.
So gelten bei der Erstellung eines SVN-Accounts keine formalisierten Eintrittskriterien, wie ein Mindestmaß technischer Fähigkeiten. Als Selektionskriterium gilt lediglich die konstante Beteiligung über einen gewissen Zeitraum. Entsprechend ist dem Autor auch kein Fall bekannt, bei dem einem Entwickler die Erstellung eines SVN-Account verwehrt wurde.

Die niedrigen Eintrittshürden machen eine klare Unterscheidung von Publikums- und Leistungsrollen im Projekt schwierig. Der Übergang von der Publikums- zur Leistungsrolle gestaltet sich fließend und setzt keine langjährige Ausbildung (wie im Fall des Lehrers, Arztes oder Anwalts) voraus. Während klassische Softwareentwicklung eine klare Trennung zwischen Nutzer (Publikumsrolle) und Entwickler (Leistungsrolle) vorsieht und besondere Stellen vorsehen muss, um das Feedback der Nutzer aufzugreifen und ggf. in Änderungen des Programms einfließen zu lassen, kann diese Feedbackschleife in Open-Source-Software-Projekten umgangen werden. Der Nutzer kann in die Leistungsrolle wechseln und Programmfehler beseitigen oder gewünschte Funktionen implementieren. Die Entwickler in Open-Source-Software-Projekten können daher als „Prosumer“ (Toffler 1980) bezeichnet werden. Der Begriff des Prosu-

mers beschreibt die Aufhebung der eindeutigen Rollendifferenzierung zwischen *Producer* und *Consumer.* Differenzierungstheoretisch (vgl. Volkmann 2010) entspricht der „Prosumer“ der sekundären Leistungsrolle, die nach Stichweh (2005:35) „eine Art aktivistischer Alternative zu einem reinen Publikumsstatus“ ist. Tatsächlich können Open-Source-Projekte dahingehend interpretiert werden, dass sie die durch die Wirtschaft angebotenen Leistungsrollen ablehnen und die Nutzer selbst die Aufgabe „Produktion“ in sekundären Leistungsrollen übernehmen.

Trotz der offensichtlich niedrigen Eintrittskriterien bilden sich in dem Projekt starke Verhaltenserwartungen aus, die an die Mitglieder gestellt werden und deren Akzeptanz unterstellt wird, wenn sich jemand im Projekt engagiert/engagieren möchte. Die (formalen) Erwartungen bei KDE setzen sich aus zwei Komponenten zusammen, dem „Code of Conduct“ und der „SVN Commit Policy“.
Im „Code of Conduct“ (vgl. KDE.org 2010a) sind Erwartungen festgehalten, die als Leitlinien der Zusammenarbeit und des Umgangs untereinander gelten sollen. Zu den im Verhaltenskodex festgehaltenen Erwartungen zählen ein respekt- und rücksichtsvoller Umgang miteinander, ein pragmatischer und kollaborativer Arbeitsstil ebenso, wie anderen Entwicklern Hilfe anzubieten aber auch Hilfe anzunehmen. In diesem Verhaltenskodex sind neben den Entwicklern explizit auch Nutzer einbezogen, da diese ebenfalls die Kommunikationsstrukturen des Projekts nutzen um Hilfestellungen bei Problemen zu bekommen, Fehler in der Software zu melden oder neue Funktionen vorzuschlagen.
„This Code of Conduct is shared by all contributors and users who engage with the KDE team and its community services. [... It] presents a summary of the shared values and ‘common sense’ thinking in our community.” (ebd.) Die starke soziale Generalisierung dieser Verhaltenserwartungen auf die Community führt dazu, dass allen Beteiligten die Zustimmung zu diesem Set von Verhaltenserwartungen unterstellt wird (vgl. Luhmann 1999: 68).

Neben diesem allgemeingehaltenen Verhaltenskodex gibt es eine „SVN Commit Policy“ (vgl. KDE.org 2010c), in der die formalen Erwartungen an die Übermittlung von Code festgehalten sind. Diese zweite Komponente der formalen Erwartungen betrifft dementsprechend nur die Mitglieder, wie wir sie definiert

haben. Zu den hier festgelegten Erwartungen zählt, dass Quellcode von anderen Mitgliedern nicht ohne deren Einverständnis verändert werden darf, dass Veröffentlichungspläne beachtet werden (insbesondere die „Freeze“-Termine, die den Zeitpunkt markieren, ab dem keine neuen Funktionen mehr in den Quellcode implementiert werden dürfen) oder dass nicht-funktionierender Code nicht übermittelt werden darf. Ebenso wird erwartet, dass Code nicht übermittelt werden darf, solange es von anderen Entwicklern Widerspruch gegen die betroffene Funktion gibt. Die vielleicht wichtigste Komponente der hier festgehaltenen Erwartungen ist die Festlegung auf eine standardisierte Benennung des Quellcodes. Auf diese Weise wird eine strukturelle Redundanz der verschiedenen Quelltexte unterschiedlicher Programmteile gewährleistet. Damit geht die Erwartung einer guten Dokumentation des Programmcodes einher. Auf diese Weise können Quelltexte von der individuellen Wissensbasis der Entwickler, die diese Texte erstellt haben, entkoppelt werden. Anderen Entwicklern ist es so ohne längere Einarbeitungsphasen möglich in den Quelltexten arbeiten zu können.

Diese beiden Komponenten konstituieren die formalen Erwartungen, die an die Rolle des Mitglieds gestellt werden und direkt auf diese Bezogen sind. In der ersten Komponente wird explizit darauf verwiesen, dass die Missachtung dieser Erwartungen negativ sanktioniert wird und zum Ausschluss, also dem Verlust der Mitgliedschaft führen kann: „Leaders of any group, such as moderators of mailing lists, IRC channels, forums, etc., will exercise the right to suspend access to any person who persistently breaks our shared Code of Conduct.” (KDE.org 2010a). Die Verhaltenserwartungen werden auch in der Zeitdimension generalisiert. Als normative Erwartungsstruktur wird im Fall von Abweichungen wird nicht durch Lernen, sondern durch negative Sanktion und Aufrechterhalten der Erwartung reagiert.

Wie gezeigt wurde existieren in dem Projekt Mechanismen mit Hilfe derer zwischen Mitgliedern (aktive Entwickler) und Nicht-Mitgliedern (Interessenten, Nutzer, etc.) unterschieden werden kann. Die beim Eintritt in den Entwicklerstatus zu akzeptierenden, formalen Erwartungen, die durch das Projekt vorausgesetzt werden, sind auf der Ebene allgemeiner Verhaltenserwartungen angesiedelt. Nicht spezifiziert werden Art und Weise oder Umfang der durch die Entwickler

zu erbringenden Leistungen. Ebenso bleiben geltende Kommunikationswege und Entscheidungsprogramme unterspezifiziert. Es kann daher festgehalten werden, dass das Projekt KDE auf der Ebene der explizierten Verhaltenserwartungen nur in einem sehr geringen Umfang formalisiert ist. Dieser Umstand erscheint Erklärungsbedürftig.

Das Projekt KDE ist, wie auch andere Open-Source-Software-Projekte, durch freiwillige und unbezahlte Arbeit der Entwickler gekennzeichnet. Statt die Entwickler für ihre Arbeit zu entlohnen ist das Projekt vielmehr auf Spenden ihrer Mitglieder (aber auch externer Geldgeber) angewiesen, um Serverstrukturen, Konferenzen und ähnliches zu finanzieren. Da keine Geldzahlungen erfolgen, kann im Projekt auch kein System „bezahlter Indifferenz“ (Luhmann 1999: 96) etabliert werden. Dies hat zur Folge, dass die Leistungsmotivation nicht von der Teilnahmemotivation abgesondert werden kann. Ein (zu) hoher Formalisierungsgrat birgt immer die Gefahr den Mitgliedern ihre Teilnahmemotivation zu entziehen. Damit entfällt für das Projekt KDE ein charakteristisches Merkmal formaler Organisationen. Einen Ansatz zur Erklärung dieses Umstands liefert Horchs (1985) Konzept „freiwilliger Vereinigungen“. Danach müssen Organisationen, die auf freiwillige und unbezahlte Mitarbeit abstellen, zur Einbindung ihrer Mitglieder Anreize schaffen, die „direkt mit dem Ziel, der Struktur, bestimmten Personen oder Gruppen der Vereinigung oder mit dem Selbst des Handelnden verbunden sind“ (ebd.: 264). Im Einzelnen sind die Mechanismen, auf die freiwillige Vereinigungen zurückgreifen können, „Prestige (Ehre), Ziel, sozialer Tausch (Anerkennung, Dank, Vertrauen, Achtung, Rang), Gemeinschaft, Identifikation und Sozialisation“ (ebd.: 263).
Jeden dieser Mechanismen findet man, teils mehr, teils weniger stark ausgeprägt, im Projekt KDE. Unter den Entwicklern ist ein hohes Maß an Identifikation mit den Zielen des Projekts und dem normativen Rahmen der „Open-Source-Bewegung“ zu beobachten. Zudem werden Entwickler durch die Möglichkeit des Wissens- und Reputationserwerbs sowie anderen Formen der Anerkennung durch Dritte zu einem längerfristigen Engagement und verbindlicher Aufgabenübernahme motiviert (vgl. Taubert 2006: 132ff.).

Dadurch, dass die Hürden für den Ein- und Austritt in das System relativ niedrig sind, sieht sich das Projekt einer ständigen Fluktuation ihrer Mitglieder ausge-

setzt. Die vergleichsweise wenig bestimmte Erwartungs- und Aufgabenstruktur erfüllt unter diesen Bedingungen die (latente) Funktion, dass die Mitglieder sich nicht sicher seien können, „wie weit ihre Pflichten gehen, und das macht sie Aufnahmebereit für neuartiges Verhalten“ (Luhmann 1999: 151). Dies stellt einen Lösungsmechanismus für Probleme dar, die aus einer nicht spezifizierten Dauer der Mitgliedschaft und damit einem raschen Wandel der internen Umwelt des Projekts erwachsen. Programmierer, die bisher einen bestimmten Programmteil betreuten scheiden aus und jemand anderes übernimmt die Betreuung, auch wenn er einst aus anderen Gründen mit der Entwicklertätigkeit begonnen hatte.

In dem Projekt ist eine eindeutige Adressierung von Erwartungen, d.h. die Zuteilung von Aufgaben nicht möglich. Stattdessen geschieht die Übernahme von Aufgaben auf freiwilliger Basis. „At this point you may be wondering who has to work on the ‚boring stuff’. A lot of software development involves fixing bugs – hardly glamorous work. Yet when you have a large enough pool of talent to draw from, you quickly find that what is tedious chore for one person is an engrossing brain-teaser for someone else. Moreover, hackers often reserve their highest praise for colleagues who take on the scut work that others shirk.” (Hamel 2007: 210)
Hamel identifiziert zwei Mechanismen, die es Open-Source-Software-Projekten ermöglichen keine defizitären Bereiche auszubilden. Zum einen ist in Open-Source-Projekten eine große Zahl von Entwicklern involviert, die es wahrscheinlich macht, dass sich jemand findet, der Spaß an genau dieser zu erledigenden Aufgabe hat. Dies steht klassischen Organisationen, in denen personale Ressourcen begrenzt sind, nur bedingt offen. Zum anderen erreichen Entwickler, die unbeliebte Aufgaben, wie das Beseitigen von Bugs[10], übernehmen, ein hohes Ansehen. Nicht zuletzt deshalb, weil diese Aufgaben einen sehr guten Überblick über bestehenden Quelltext, sowie die Interdependenzen zwischen verschiedenen Programmteilen erfordern.

Während Mitglieder in Luhmanns Konzept formaler Organisationen nur ein Mittel zum Zweck der Anpassung an eine „ausgewählte Umwelt von Nicht-

[10] Unter dem Begriff „Bug“ werden Programmfehler verstanden.

Mitgliedern“ (Horch 1985: 260) sind, verhält es sich in freiwilligen Vereinigungen gerade umgekehrt: „Hier setzen die Mitglieder die Ziele. Die Beziehungen zur Außenumwelt sind Mittel zum Zweck der Verfolgung der Mitgliederinteressen.“ (ebd.) Gerade deshalb können freiwillige Vereinigungen Entscheidungen in einem viel geringeren Maß vorstrukturieren und vorgeben, als es in klassischen Organisationen der Fall ist. Sie sind auf eine „demokratische Entscheidungsstruktur“ (vgl. ebd.: 261) angewiesen, in der den Mitgliedern die Chance eingeräumt wird, auf Ziele des Projekts Einfluss zu nehmen.

5.2 Entscheidungsprämissen im Projekt

> *“Your actions and work will affect and be used by other people and you in turn will depend on the work and actions of others. Any decision you take will affect other community members, and we expect you to take those consequences into account when making decisions.”* (KDE.org 2010a)

Während die Grenzziehung über die Zurechnung von Kommunikation auf Mitglieder erfolgt, werden Strukturen des Systems durch eine selektive Kombination von Ereignissen aufgebaut. Die Systembildung wird also erst „durch besondere Arten von Ereignissen ermöglicht“ (Luhmann 1980: 243). Im Fall von Organisationen sind diese Ereignisse Entscheidungen. Entscheidungen sind hier nicht als „ein psychischer Vorgang“, sondern eben als „soziales Ereignis“ – als Kommunikation – zu verstehen (Luhmann 1988: 166). In Organisationen ist Entscheidung nicht eine unter anderen möglichen Kommunikationsformen, sondern vielmehr die einzig mögliche. So kann „das Handeln, und zwar jedes Handeln, im System als Entscheidung behandelt werden“ (ebd.: 166). Dadurch kann das System seine selbstgezogenen Grenzen reproduzieren und sich selbst als unterschieden von der Umwelt, als eigene Sinnsphäre beobachten. Als Ereignisse verschwinden Entscheidungen im Moment ihrer Entstehung wieder. Sie markieren eine „Diskontinuität, also eine Differenz von vorher und nachher“ (ebd.: 169). Auf der Ebene der Elemente sind Organisationen daher „nicht bestandsfähig“ (ebd.: 168). Erst durch die selektive Verknüpfung von Ereignissen mit Ereignissen können Systeme bestandsfähige Strukturen ausbilden. Das Bestandsproblem von Organisationen liegt daher nicht in der Erhaltung organisationaler Strukturen, sondern in der Reproduktion von Entschei-

dungen.
Organisationen sind autopoietische Systeme, „die aus Entscheidungen bestehen und die Entscheidungen, aus denen sie bestehen, durch die Entscheidungen aus denen sie bestehen, selbst anfertigen“ (ebd.: Herv. i. Org.). Wie andere Ereignisse auch, transformieren Entscheidungen Kontingenz und bringen diese in eine fixierte Form. Dies entfaltet die Paradoxie, dass die Entscheidung vor der Entscheidung eine andere ist als nach der Entscheidung. Nach der Entscheidung eröffnet sie einen Bereich neuer Entscheidungsmöglichkeiten und wird anschlussfähig. Was auf eine Entscheidung folgt, ist jedoch nicht beliebig. Die Möglichkeiten werden durch Erwartungsstrukturen eingeschränkt.

Als Struktur dient einem System alles, was „die Distanz zwischen Entscheidung zu Entscheidung“ überbrückt (ebd.: 172). Auch für diese Überbrückungsleistungen nehmen Entscheidungen eine exponierte Rolle ein. So gibt es „Entscheidungen, die Entscheidungsprämissen für eine noch unbestimmte Vielzahl anderer Entscheidungen festlegen. In solchen Fällen wird die Reichweite von Entscheidungen durch Einschränkung ausgedehnt; oder [...] Komplexität durch Reduktion erzeugt“ (Luhmann 2006: 223 - Herv. i. Org.). Strukturaufbau dient dann dazu, das erwartbar zu machen, was Folgen kann. Nach Luhmann existieren vier Typen solcher Prämissen: Konditional- und Zweckprogramme, Festlegung von Kommunikationswegen sowie Personen (vgl. Luhmann 1988: 176ff.). Im Folgenden sollen Konditional- und Zweckprogramme, sowie Personen betrachtet werden. Der Regulierung von Kommunikationswegen widmet sich das nächste Kapitel.

Konditionalprogrammierungen stellen Strukturen mit einem *wenn-dann-Zusammenhang* dar. Diese Form der Strukturierung von Entscheidungszusammenhängen tritt im Projekt KDE vor allem in zwei Kontexten auf: zum einen in Form von Fristen, zum anderen in der Form von Kompabilitätserwartungen. Die Achtung der Fristen ist ein Bestandteil der SVN-Commit-Policy und damit bindend für alle Entwickler. Fristen sind im Projekt festgelegte Termine, die der Veröffentlichung einer neuen Version der Software vorausgehen. Hier findet sich eine wiederkehrende Abfolge von Phasen, die regulieren, ab wann der Software einer bestimmten Versionsnummer keine neuen Funktionen mehr hinzugefügt werden dürfen („Freeze“) und sich die Entwicklung auf die Beseitigung

von Fehlern beschränkt. Dabei ist die Entwicklung neuer Funktionen in diesen Zeiträumen keineswegs Ausgeschlossen. Die Entwicklungen neuer Programmbestandteile finden dann nur unter einer neuen Versionsnummer statt, deren Veröffentlichung nicht unmittelbar bevor steht.
Der zweite Kontext, in dem Konditionalprogrammierung von Bedeutung ist, ist der Bereich der Kompatibilität. Auch die Kompatibilitätserwartungen sind in der SVN-Commit-Policy festgehaltene Verhaltenserwartungen. Von Kompatibilität wird im Zusammenhang der Softwareentwicklung gesprochen, wenn neu hinzugefügter Quelltext nicht zu Fehlern und Funktionsstörungen in bereits existierenden Bestandteilen führt. Wenn solche Störungen auftreten, darf der entsprechende Quelltext nicht implementiert werden und muss einer gründlichen Überarbeitung unterzogen oder verworfen werden. Dadurch kann sich die Implementation einer bestimmten Funktion teilweise über mehrere Versionsveröffentlichungen hinziehen, bis der Code so stabil ist, dass er keine Brüche mit der bestehenden Codebasis verursacht. Bei solch großen, zeitintensiven Änderungen erfolgt die Entwicklung in sog. „Branches“. Dies sind Auskopplungen aus der eigentlichen Quelltextbasis, in denen neue Funktionen implementiert werden, ohne dass die Stabilität der regulären Quelltextbasis gefährdet wird.
Eine Ausnahme von dieser Regel stellen Versionssprünge dar. Im Fall eines Versionssprungs wird die bestehende Codebasis verworfen und neu programmiert. Es ist eine Situation, über die Karl Weick (1996) schreibt: „Dropping one’s tools is a proxy for unlearning, for adaption, for exibility“. Im Fall des Projekts KDE ist dies zuletzt mit dem Versionssprung auf 4.0 geschehen (vgl. Kügler 2008). Auslöser dieser Diskontinuität waren neue Möglichkeiten in der Oberflächenbibliothek QT, auf die KDE aufbaut. In der Folge wurden in diesem Zustand totaler Offenheit innerhalb kürzester Zeit viele Innovationen und neue Technologien entwickelt.

Zweckprogrammierung erfolgt über die Festlegung eines gewissen Outputs, wobei die Mittel, die zur Erreichung verwendet werden, offen bleiben. Diese Form der Programmierung ist die wahrscheinlich am meisten verwendete Form im Projekt KDE. Die auf den Mailinglisten diskutierten, selbstgesetzten Ziele werden weitgehend frei von Einschränkungen verfolgt. Kriterium ist dabei nicht der bestmögliche Output, sondern das Kriterium des „satisficing“ (Bonazzi 2008:

282 - Herv. i. Org.), das dem oben genannten Anspruch an Kompatibilität entspricht. Ausgehend von dieser Basis erfolgt dann eine Vielzahl kleinschrittiger Verbesserungen des Quelltextes - ganz nach dem Mantra der Open-Source-Entwicklerszene: „release early, release often“. Quelltexte werden dabei im Sinne des *rapid prototyping* in einem sehr frühen Stadium veröffentlicht, um diese anschließend im Dialog mit anderen Entwicklern, inkrementell zu verbessern.

Da das Projekt nur zu einem geringen Maß formalisiert ist, liegt die Vermutung nahe, dass Personen einen besonders hohen Stellenwert in den Projekten einnehmen. Und tatsächlich ist KDE in hohem Maße auf nicht festlegbare Entscheidungskompetenzen und Wissensbestände seiner Mitglieder angewiesen. Dies birgt die Gefahr, dass sich im Projekt durch Wissen über Quelltexte oder Kontrolle von relevanten Umweltbeziehungen Machtkonstellationen ausbilden können. Hier greifen jedoch strukturelle Vorkehrungen im Projekt und schließen eine Verwendung von Wissen zum Ausbau von Macht relativ wirksam aus. Das in die Quelltexte einfließende Wissen ist auf Grund der Offenheit der Quelltexte immer einsehbares und dokumentiertes Wissen, das nicht verloren geht, selbst wenn ein Entwickler das Projekt verlässt. Ebenso wichtig für diesen Aspekt sind die Mailinglisten. Eine wichtige Funktion der Mailinglisten besteht darin, dass sie das „Gedächtnis“ des Systems konstituieren. Die gesamte, über die Listen sattfindende Kommunikation wird archiviert. Damit kann zu jeder Zeit und von Jedem in den Archiven recherchiert werden, wer, was, wann, auf welche Weise gelöst hat. Die Entwicklung wird so weitgehend von den Einzelbewusstseins und Wissensständen der involvierten Akteure entkoppelt. Durch das Lizensierungsmodell ist außerdem ausgeschlossen, dass ein Entwickler im Fall seines Ausscheidens die von ihm entwickelten Quelltexte „mitnimmt“. Auf veröffentlichten Quellcode können keine Besitzansprüche erhoben werden. Auch Umweltbeziehungen scheiden weitgehend als Quelle der Macht aus. Beziehungen zu anderen Projekten und Organisationen sind nie an eine Person gebunden. Gerade weil das Projekt KDE derart dezentral operiert und nicht von einzelnen Stellen aus gesteuert werden kann, bestehen Beziehungen immer an vielen Schnittstellen des Projekts. Daher scheint auch hier die Möglichkeiten begrenzt

zu sein, Beziehungen für einen Ausbau einer Machtposition im Projekt zu nutzen.

5.3 Kommunikationswege und die Rolle des Maintainers

Das Projekt KDE ist durch die nicht-hierarchische Kommunikationsmedien, wie Mailinglisten und Blogs geprägt. Eine Festlegung an wen welche Information weitergegeben werden muss, ist in diesen Medien nur schwer zu etablieren. Jede Information ist für jeden einsehbar. Informationsbedürfnisse sind daher immer an eine proaktive Suche nach eben jenen Informationen gebunden. Dies gilt auch für die Betreuer einzelner Subprojekte, die „Maintainer“ und führt zu einer entscheidenden Prägung ihrer Funktion.
Maintainer sind verantwortlich für ein (Sub-)Projekt. Sie sind jedoch keine „Chefs“ im klassischen Sinne. Die Vergabe dieser exponierten Rollen erfolgt auf der Grundlage von Reputation. Diese können Akteure auf Grund ihrer technischen Fähigkeiten, sowie ihrer Beteiligungshistorie erwerben. Während Macht ihren Ursprung in der Sozialdimension hat, konstituiert sich Reputation durch Differenzen in Sach- (technische Fähigkeiten) und Zeitdimension (Projektzugehörigkeit). Durch Reputation werden Personen zu „relevanten Anderen“, mit denen im Entwicklungsprozess Einverständnis erzielt werden muss. Dieser Umstand kann jedoch nicht als Machtverhältnis in einem instrumentellen Sinn verstanden werden, vor allem da an Reputation keinerlei Sanktionsmöglickeiten gebunden sind. Vielmehr besteht ein Zusammenhang zwischen Reputation und den Einflussmöglichkeiten, die ein Akteur auf den Entwicklungsprozess hat.

Die Rolle des Maintainers wird von einem aktiven KDE-Entwickler wie folgt beschrieben: „Maintainer bedeutet bei uns nichtwirklich irgendwer mit Macht über irgendwelche Quelltext-Bereiche, sondern das sind Leute, die haben halt in nem gewissen Bereich nen Überblick“ (Transkipt: 6ff.). Top-Down Entscheidungen im klassischen Sinn treffen sie nicht. Stattdessen werden anstehende Entscheidungen über das Medium „Mailingliste“ mit allen Mitgliedern diskutiert „und dann kommt man am Ende zu nem Konsens, der gut für alle ist“ (ebd.: 26f.). Ähnlich beschreibt auch Linus Torvalds, Maintainer des Linux-Kernels und „Begründer“ dieses Rollenverständnisses, seine Aufgabe: „Ich versuche zu mana-

gen, in dem ich keine Entscheidungen treffe und den Dingen ihren Lauf lasse. So bekommst du die besten Ergebnisse“. (zitiert nach Picot/Fiedler 2008: 239).

Die Rolle impliziert nicht die Durchsetzbarkeit eigener Vorstellungen. Maintainer können keine formal bindenden Entscheidungen treffen, sondern müssen die Auseinandersetzung mit anderen Mitgliedern auf der Mailingliste suchen. Als Entwickler mit hohem technischen Verständnis und hoher Reputation können sie zwar oftmals bessere technische Argumente liefern. Gleichzeitig ist es so, dass sich auch „die kompetenten Reputationsträger nicht immer durchsetzen bzw. niemanden zwingen können, ihre Meinung zu übernehmen“ (Brandt 2009: 113).

Vielmehr sind auch die Maintainer darauf angewiesen Konsens für ihre Positionen zu erzeugen. Denn: die Sonderrolle des Maintainers ist jederzeit vakant und dauert nur so lange an, wie die anderen Mitglieder des Projekts dem Maintainer seine Sonderrolle zugestehen. Der Managementtheoretiker Gary Hamel schreibt über die Hierarchien in Open-Source-Software-Projekten: „Hierarchies get built from the bottom up, not from the top down. In that sense they are ‚natural’ rather than ‚proscribed‘“(Hamel 2007: 205). Hierarchien bilden sich also informell und sind nicht durch die formale Struktur des Projekts gedeckt.
Der Schlüssel zur herausgehobenen Bedeutung von Konsens in Open-Source-Projekten liegt in der „forkability“ (Fogel 2007:56) der Projekte. D.h. jedem steht es offen, den existierenden Code zu nehmen und ein konkurrierendes Projekt zu initiieren. „The paradoxical thing is that the possibility of forks is usually a much greater force in free software projects than actual forks, which are very rare.” (ebd.)
Die Möglichkeit den Projektcode zu teilen, führt dazu, dass niemand *Macht* über den Quellcode – und damit auch keine Macht über andere Ressourcen im Projekt – beanspruchen kann. „A key property of all open source licenses is that they do not give one party more power than any other in deciding how the code can be changed or used.” (ebd.)

Dieser weitere zentrale Unterschied zur klassischen Hierarchie besteht darin, dass den Maintainern die Möglichkeit fehlt Entscheidungen über die Distribution von Ressourcen zu treffen. Sie können den Entwicklungsprozess also nicht

steuern, indem sie einzelnen Entwicklungszielen mehr Geld oder Personal zuweisen als Anderen. Dies hängt damit zusammen, dass dem Projekt so gut wie keine monetären Mittel zur Verfügung stehen, mit deren Hilfe einzelne Entwicklungsziele vorangetrieben werden könnten. Humane Ressourcen hingegen können durch das geringe Maß an Formalisierung, sowie fehlende Weisungsbefugnisse und Sanktionsmöglichkeiten, nicht zu einzelnen Subprojekten zugeteilt werden. Die Entwickler finden sich in Prozessen der Selbstorganisation zusammen. Und dieser Prozess, in den ständig neue Entwickler hinzukommen und Beiträge liefern, andere der weiteren Entwicklung jedoch fern bleiben, kann und wird nicht von einer zentralen Stelle koordiniert und kontrolliert.
Der einzig (theoretisch) zur Verfügung stehende Sanktionsmechanismus besteht im Ausschluss einzelner Projektmitglieder aus dem Projekt. Dies wäre zwar technisch möglich, wird jedoch nicht genutzt. Es würde der allgemeinen Norm des freien Informationsflusses, sowie dem partizipierenden Charakter des Projekts widersprechen. Zudem würde dem Maintainer wahrscheinlich die Folgschaft der anderen Mitglieder versagt, wenn kein breiter Konsens über den Ausschluss bestünde. Außerdem würde ein solcher Ausschluss vermutlich einen „Fork“ des Projekts nach sich ziehen, was mit Nachteilen für das gesamte Projekt verbunden wäre.

Als Mechanismus zur Reduktion von Unsicherheit dient die im Projekt vorzufindende Form der Hierarchie daher nur sehr bedingt. Schließlich sind formale Autorität und faktische Macht stark begrenzt.
Stattdessen bedient die Rolle des Maintainers eher Grenzstellen-Funktionen nach Außen, sowie jene Expertenfunktion nach Innen, in der der Maintainer anderen Entwicklern bei technischen Problemen hilft und das Projekt durch gute technische Argumente leiten kann. Zudem wird durch den Maintainer auch ein gewisses Maß an Stabilität und Redundanz in das Projekt eingeführt.
Die Maintainer stellen eine, an die Person gebundene, Konstante in der Entwicklung dar. Sie verfügen über einen besseren Überblick über den Quellcode und investieren mehr (Arbeits-) Zeit in das Projekt, als die meisten anderen Entwickler. Dies, gepaart mit dem hohen Maß an Reputation, mit dem die Maintainer ausgestattet sind, führt zu einer erhöhten Wahrscheinlichkeit der Durchsetzung ihrer eigenen Vorstellungen und Ziele. Sie haben „Einfluss“ (vgl. Zün-

dorf 1986: 37ff.). Dabei sind sie darauf angewiesen ihre Ziele argumentativ darzulegen und andere Entwickler von ihren Zielen zu überzeugen. Ihre Einflussmöglichkeiten müssen demnach auf die „Verständigung“ (vgl. ebd.: 43ff.) mit anderen Entwicklern abzielen. Die herausgehobene Stellung der Maintainer beruht nicht auf formaler Autorität, wie es in Hierarchien formaler Organisation vorzufinden ist, sondern auf der symbolischen Ressource der „Reputation“ aus welcher die gesteigerten Einflusschancen auf Entscheidungsprozesse resultieren.

5.4 Entscheidungsprozesse

Mailinglistenkommunikation dient der Entscheidungsfindung, der Verständigung auf Entwicklungsziele und der Koordination bei der Verfolgung eben jener Ziele. Brandt (2009: 112) schreibt über die digitale Kommunikationskultur, dass „Ziele [...] ohne Ansehen der Person ausdiskutiert“ werden und der „zwanglose Zwang eines besseren Arguments“ herrsche. Unter den Entwicklern sei zudem ein „kooperative[r] Arbeitsstil“ sowie eine „konzeptionelle Aufgeschlossenheit“ (Hoffmann 1998: o.S.) zu beobachten.

Die Kommunikation durch die Mailingliste erfolgt im Modus der Argumentation. „Eine Einflußnahme auf Entwicklungsziele ist nur möglich, sofern der betreffende Akteur seine Position begründet“ (Taubert 2006: 163). Dies gilt für einfache Entwickler ebenso wie für die Maintainer. Statt Top-Down-Entscheidungen zu fällen, müssen auch sie ihre Standpunkte argumentativ vertreten. Dies bedeutet auch, dass sie nicht auf ihre Sonderrolle, Beziehungen zu anderen Entwicklern oder bereits erbrachte Leistungen verweisen können.

Um ein Entwicklungsziel zu verfolgen muss in einer Mailinglistendiskussion kein Konsens zwischen den Entwicklern hergestellt werden. Stattdessen reicht „die Herstellung einer Situation, in der sich kein weiterer Widerstand gegen die vorgeschlagenen Entwicklungsziele formiert.“ (ebd.: 170 f.) Die Entscheidung muss demnach keine optimale, sondern lediglich eine befriedigende Situation herstellen (vgl. Berger/Bernhard-Mehlich 2006: 212). Die Diskussion zielt auf Verständigung zwischen den beteiligten Akteuren. Entwickler, die durch ihre Beteiligungsgeschichte im Projekt Reputation erworben haben, haben im dargestell-

ten Rahmen größere Möglichkeiten der Einflussnahme, als reputationsarme Entwickler.

Das Projekt ist auf ein hohes Maß an Korrelation zwischen den Zielen des Projekts und den individuellen Zielen einzelner Entwickler angewiesen. Dies bedingt eine hohe strukturelle Aufgeschlossenheit gegenüber Neuem, da mit dem Hinzutreten neuer Entwickler immer auch neue Ziele evoziert werden, deren Unterbindung mit der Gefahr verbunden wäre, diese Entwickler wieder zu verlieren. Dabei kommt auch hier der Nutzen einer geringen Fomalstruktur des Projekts zum Vorschein. Denn die schnelle Anpassung an immer neu gesetzte und schnell veränderliche Ziele ist nur durch eine geringe Formalisierung und die „ambivalente Unklarheit von Aufgabenzuweisungen und Verantwortlichkeiten“ (Luhmann 1999: 151) möglich.
Neue Entwicklungsziele bedeuten immer einen Aushandlungsprozess zwischen den beteiligten Akteuren, in dem, strukturell verankerter Aufgeschlossenheit zum Trotz, Ablehnung möglich ist. Nun könnte man annehmen, dass sich in diesen konfliktträchtigen Diskussionen um Entwicklungsziele Positionen verhärten und die weitere Entwicklung dadurch blockiert würde.

Dieses Phänomen tritt jedoch faktisch nicht auf. Taubert (2006: 174) zitiert einen Entwickler mit den Worten: „Irgendwann sind dann die Leute angenervt genug, daß sie sich dann zusammenraufen und dann irgendwas zusammenbauen.“ Es lässt sich das Phänomen der „Diskussionsmüdigkeit“ (ebd.) beobachten, dass darauf zurückzuführen ist, dass die Programmierer eher an einer aktiven Entwicklung als an andauernden Diskussionen interessiert sind. Diese, auf den Akteur zielende Erklärung, kann um eine weitere, strukturell verankerte Form der Konfliktlösung erweitert werden: die Konfigurationsoption. Diskussionen auf Mailinglisten können an einen Punkt gelangen, an dem kein Kompromiss zwischen den Parteien möglich ist. Vorschläge für neue Programmfunktionen argumentieren häufig mit grundlegenden Bewertungsnormen der Open-Source-Bewegung. Dabei setzt sich das Projekt der Gefahr der Wertekonflikte aus, in denen sich zwei, von allen Entwicklern prinzipiell geteilte, Werte unvereinbar gegenüberstehen. So ist die Verfolgung eines Werts, wie der einfachen Anwendbarkeit des Programms, nur auf Kosten der Einschränkung der Handlungsfreiheit, also Reduzierung von Optionen möglich. In einer solchen

Konstellation wird dann häufig auf die, in diesem Fall Paradox wirkende, Lösung der Konfigurationsoption zurückgegriffen. Die Lösung läge darin, dass die Standartkonfiguration einer einfachen Benutzbarkeit folgt, weitere Funktionalitäten jedoch über ein Menü aktiviert werden können.
An diesem strukturellen Ausweg aus der Konfliktsituation wird eine bereits angesprochene Besonderheit der Mailinglistenkommunikation deutlich. Ziel dieser Diskussionen ist nicht die Herstellung eines Konsenses zwischen allen Entwicklern, sondern viel mehr die Abwesenheit offenen Protests. Der Einbau von Konfigurationsoptionen erlaubt es den inkonsistenten Zielsetzungen der Entwickler zu Folgen. Konfigurationsoptionen kommen somit den „Wünschen sämtlicher Beteiligte[n] entgegen“, indem sie sowohl „dem Wunsch nach Verfügbarkeit, als auch nach Abwesenheit der Funktionalität Rechnung tr[agen]“ (ebd.: 171).

Nicht jedes Entwicklungsziel muss über die Mailinglisten diskutiert werden. Können Entwickler ein hohes Maß an Konsens über das angestrebte Ziel unterstellen, steht es ihnen offen eine Funktion zu implementieren, ohne zuvor die Diskussion mit den anderen Entwicklern gesucht zu haben. Durch die Konsensunterstellung können kleinere Entwicklungsschritte enorm beschleunig und der kommunikations- und abstimmungsbedarf reduziert werden. Ein Fall, in dem breiter Konsens unterstellt werden kann, ist z.B. die Beseitigung eines Bugs. Entwickler können unterstellen, dass auch anderen Entwicklern daran gelegen ist, dass der Fehler aus dem Programmcode entfernt wird und können diesen somit ohne vorherige Diskussion entfernen.
Da es möglich ist Programmcode ohne vorangegangene Diskussion zu verändern, treten immer wieder Situationen auf, in denen die Konsensunterstellung fehlschlägt. Ein Entwickler hatte angenommen, dass es einen breiten Konsens für die von ihm implementierte Funktion gäbe – nun regt sich jedoch Widerstand dagegen. Interessant an diesen Fällen ist, dass die Mailinglistenkommunikation diese Fälle dann so behandelt, als seien sie noch nicht implementiert. Möglich ist dies durch die Versionsverwaltungssoftware, mit der bereits implementierte Codebestandteile wieder aus dem Quellcode entfernt werden können, ohne dadurch andere nach dem jeweiligen Beitrag veränderte Codebestandteile in Mitleidenschaft zu ziehen. So kann dieser Beitrag von den Entwicklern behandelt werden, als sei er noch gar nicht getätigt worden, sodass sich der Konflikt

nicht auch in die Sozialdimension ausbreitet, sondern auf die Sachdimension der technischen Argumentation beschränkt bleibt.

5.5 Folgeprobleme

Eine der Folgen dieser Struktur ist die Zunahme von Koordinationsbedarf und damit Kommunikation in dem Projekt. Es bildet sich ein „neues kommunikationsorientiertes Arbeitsverständnis“ (Kühl 1998: 63) heraus. So stellt das Projekt eine Vielzahl von Kommunikationsmedien zur Verfügung, die alle hochfrequentiert sind. So gibt es durchschnittlich 39,4 Blogeinträge pro Woche von Entwicklern, die sich mit Neuerungen, Entwicklungen und Ideen zum Thema KDE befassen.[11] Mailinglisten weisen eine noch höhere Frequenz auf. Die Mailingliste „KDE-Core-Devel“ weist für das vergangene Jahr (2010) Jahr eine durchschnittliche Zahl von 454 Mails/Monat auf. Insgesamt existieren 149 zum Projekt gehörige Mailinglisten. Hierbei müssen die Entwickler selektieren, „welche Sachen für einen interessant sind und wie viel Zeit man für Mails lesen aufwenden will“ (Transktipt: 51f.). Hier werden Strategien erforderlich, um die Masse an Informationen auf die individuellen Interessen und Projektrelevanzen einzugrenzen. Damit ist bereits ein erstes Folgeproblem angesprochen: Umgang mit Informationen.

Ein Zweites ergibt sich aus der Ausübung von Hierarchien in Open Source-Projekten. Diese unterscheidet sich deutlich von der in klassischen Organisationen. Die quasi fehlende Hierarchie beeinflusst die gesamte Struktur des Projekts. So fehlt Beispielsweise eine zentrale Instanz, die eine Zuteilung von Ressourcen auf einzelne Subprojekte vornehmen könnte. Daraus entsteht die Problematik, dass einzelne Programme „verwaisen“ können, wenn ein Entwickler, der den Hauptteil der Entwicklungsarbeit übernommen hat, das Projekt verlässt. Zwar finden sich oftmals andere Entwickler, die die entstehende Lücke füllen, dies kann jedoch nicht durch das Projekt beeinflusst werden.

[11] Einsehbar unter: www.planetkde.org

6. Ein Organisationstyp für die „next society“?

> *„To compete against OSS, [Microsoft] must target a process rather than a company.”* – Vinod Valloppilli (Microsoft Entwickler) zitiert nach Hamel 2007: 208)

Das vorangegangene Kapitel hat sich vor allem mit der Beschreibung der Strukturen befasst, die sich im Open-Source-Software-Projekt KDE identifizieren lassen. Das folgende Kapitel soll diese Strukturen in einen weiteren theoretischen Rahmen einordnen und einige allgemeine Aussagen zu den gefundenen Strukturen treffen.

Das Projekt KDE wird durch drei strukturelle Besonderheiten geprägt. Zuerst kann nicht in einem klassischen Sinn von „Mitgliedschaft“ gesprochen werden, da der Zugang zu den Mitgliedschaftsrollen im Projekt (Entwicklerstatus) kaum begrenzt ist. Die Entscheidung in den Entwicklerstatus einzutreten wird faktisch nur vom Akteur, nicht jedoch vom System getroffen. Dieser Prozess der freiwilligen Selbstrekrutierung wird in erster Linie intrinsisch, durch den „Spaß am selbstbestimmten Programmieren“ (Brandt 2009: 96) motiviert. Extrinsische Faktoren spielen nur eine untergeordnete oder gar keine Rolle.

An die Mitglieder der Projekte werden nur relativ allgemeine Verhaltenserwartungen gestellt und die Akzeptanz allgemeiner Werte und Normen der Open-Source-Bewegung unterstellt. Die geringe Formalisierung der Mitgliedsrolle und die dadurch bedingte Unterdeterminiertheit der Erwartungszusammenhänge, bedingen, dass in dem Projekt vor allem die Entscheidungsprämisse „Person“ an Relevanz gewinnt. Das Projekt ist in sehr hohem Maße auf im Voraus nicht festgelegte Urteile seiner Mitglieder angewiesen.

Jedem Mitglied steht die Option offen, auf Entwicklungsziele Einfluss zu nehmen und eigene Lösungsstrategien anzubieten. Dies wird durch die „zwanglose“ Struktur des wenig formalisierten Projekts unterstützt. Diese eröffnet den Mitgliedern immer neue Freiräume, in denen sie Ideen ausprobieren und umsetzen können. Begünstigt wird dies zudem dadurch, dass sich im Projekt kein „System bezahlter Indifferenz“ ausbilden kann. Die Entwickler sind nicht extrinsisch durch Geldzahlung motiviert, sondern ziehen gerade aus den weitreichenden Möglichkeiten, die sich im Projekt bieten, ihre Bereitschaft neue Ziele einzubringen und Zeit und Energie für deren Umsetzung zu investieren.

Damit leiten wir bereits auf die Zweite strukturelle Besonderheit von KDE über. Diese liegt in der spezifischen Form der Handlungskoordination. Während klassische Organisationen aller anderslautenden Selbstbeschreibungen zum Trotz von hierarchischen Weisungsbefugnissen geprägt sind, wird das Utopia vieler Organisationstheoretiker und -berater, Enthierarchisierung der Strukturen, in dem hier untersuchten Open-Source-Projekt (fast) zur Realität. Eine klassische Hierarchie gibt es in den Projekten nicht. Zwar existieren exponierte Rollen, wie die des Maintainers, jedoch gehen mit diesen Rollen keine Weisungsbefugnisse gegenüber anderen Entwicklern oder die Kontrolle über Ressourcen einher. Sie verfügen aber über ein vergleichbar hohes Maß an Einfluss auf Entwicklungsziele. Ihren Einfluss verdanken sie ihrer Beteiligungshistorie, sowie ihren besonderen Fähigkeiten und Wissensbeständen und der Rolle.

Trotzdem lässt sich eine Effektverstärkung hin zu noch größerem Einfluss beobachten. Denn durch die Rolle werden die ohnehin reputationsreichen Entwickler zu „relevanten Anderen“ auf deren Einverständnis es im Entwicklungsprozess ankommt. Das hohe Maß an Einfluss ist jedoch äußert labil. “In the Linux sodality, power is granted from below [... and ...] is easily lost [...]. Finally the exercise of authority is constrained by the necessity for consultation and transparency.” (Hamel 2007: 207 -Herv. i. Org.) Auch in diesen Rollen stehen die Entwickler unter der Anforderung, ihre Entscheidungen öffentlich darzulegen und argumentativ zu begründen. Dabei fällt auf, dass die Diskussionen sehr sachlich verlaufen und die Entwickler eine hohe Aufgeschlossenheit gegenüber Ideen und Visionen zeigen. Die Struktur des Projekts ist, wenn auch nicht durch das gänzliche Fehlen von Hierarchien, Statusunterschieden oder unterschiedlichen Einflusschancen, doch von weitgehender Gleichheit der Mitglieder geprägt.

Ein Mantra „postbürokratischer Organisationen“ (Kühl 1998) ist Dezentralisierung. Damit kommen wir zu der dritten strukturellen Besonderheit des Projekts KDE: den verwendeten Kommunikationsmedien. Die Kommunikation im Projekt kommt weitgehend ohne face-to-face-Interaktion aus und erfolgt fast ausschließlich über Kanäle des Massenmediums Internet. Besonders frequentiert sind dabei Mailinglisten, Blogs und Chats, wobei die Mailinglisten das für die Diskussion von Entwicklungszielen maßgebliche Medium darstellen. Über die

elektronischen Verbreitungsmedien kann eine sehr hohe Anzahl an Entwicklern kommunikativ erreicht und in die Zwecke des Projekts eingebunden werden. Im virtuellen Raum der Entwicklungstätigkeit wird eine dezentrale und asynchrone Arbeitsweise möglich.
Jede über diese Kanäle erfolgende Mitteilung hat einen im Voraus unbestimmten Empfängerkreis. Dadurch ergeben sich für die Entwickler weitreichende Möglichkeiten immer neuer Formen der Selbstorganisation in Form von Ad-hoc-Teams, die gemeinsam an der Erstellung einzelner Funktionen arbeiten. Der Kreis der Entwickler kann sich dabei ständig erweitern, aber auch reduzieren. Zudem ist die Entwicklung in einem Subprojekt nur lose an Entwicklungen in anderen Subprojekten gekoppelt. Bezahlt wird diese strukturelle Flexibilität durch einen sehr hohen Kommunikationsaufwand. Hier kommt dem Projekt die funktionssystemspezifische Präferenz für Neuheit zu Gute. Die verwendeten Kommunikationsmedien und damit auch die Art, wie sich das Projekt selbst beobachtet, wirkt nicht strukturkonservierend, sondern reizt „sich selbst zu ständiger Innovation. Sie erzeugt ‚Probleme', die ‚Lösungen' erfordern, die ‚Probleme' erzeugen, die ‚Lösungen' erfordern." (Luhmann 2009: 97)

Die drei herausgestellten strukturellen Besonderheiten des Projekts KDE finden sich ebenfalls als Merkmale in dem von Henry Mintzberg (1998) beschriebenen Strukturtyp der Adhocracy. Auch in diesem Strukturtyp finden sich kaum konstante und starre Strukturen. Vielmehr werden diese „ständig neu auf die sich ändernden Anforderungen ausgerichtet." (Kieser 2006: 243). Auch KDE weist eine hohe strukturelle Flexibilität auf, um sich an neue Entwicklungsziele anzupassen. Die Struktur ist hochgradig fluid und durch eine ständige Auflösung und Rekombinierung geprägt. Eine weitere Parallele besteht in dem Einfluss, den die Mitglieder der Adhocracy auf die Formulierung von Zielen und Strategien haben. Im Projekt KDE kann jedes Mitglied eigene Ziele und Wünsche einbringen und an diesen Arbeiten. Daraus folgt jedoch auch, dass die Entwicklungen nicht von einer zentralen Stelle aus gesteuert werden können. Stattdessen wird die Verantwortung über die einzelnen Bereiche an diejenigen übertragen, die aktuell daran arbeiten. „That means power over [...] decisions and actions is distributed to various places and at various levels according to the needs of the particular issue." (Mintzberg 1998: 681)

Der Strukturtyp der Adhocracy findet vor allem in auf Innovation ausgerichteten Prozessen Verwendung, wenn die Möglichkeit zur Standardisierung der Arbeitsprozesse gering ausfällt. Dies ist gerade in der Entwicklung einer Computersoftware der Fall, bei der die Entwicklung durch ein kontinuierliches Voranschreiten geprägt ist und nicht durch das Erhalten eines einmal erreichten Status-quos. Insofern wirkt die Organisationsstruktur unterstützend bei der Umsetzung dieser Zielsetzung. Bei der Beantwortung der Frage, wieso in einer solch labilen Struktur dennoch zielgerichtetes Handeln möglich ist, nimmt die Einbettung des Projekts in die Open-Source-„Bewegung“ eine zentrale Stellung ein. Hier sind starke Überzeugungen und Werte präsent, die von allen Entwicklern, die an den Projekten partizipieren, geteilt werden. Man könnte sagen, dass diese Projekte eine starke „Unternehmenskultur“ entwickelt haben, die integrierend wirkt und dem Projekt, bei aller Flexibilität, die notwendige Stabilität liefert. Diese Kultur basiert auf gemeinsam geteilten Werten, wie dem freien Zugang zu und der freien Verfügbarkeit von Informationen.

Das Anliegen der vorliegenden Arbeit war es, die Strukturen des Open-Source-Software-Projekts KDE im Rahmen eines organisationssoziologischen Vokabulars zu beschreiben und den Blick auf die strukturellen Differenzen zu klassischen Organisationen zu öffnen. Deutlich wurde, dass es sich um einen Strukturtyp handelt, der sehr hohe Flexibilität ermöglicht. Erkauft wird diese Flexibilität durch eine sehr hohe Labilität der Strukturen, die durch eine hohe intrinsische Motivation der Entwickler und eine starke normative Fundierung des Projekts kompensiert wird. Das Projekt ist weniger statische Organisation, als vielmehr ein ständiger Prozess des (Re-)Organisieren.
Im Projekt wird die Strukturform der Adhocracy aufgegriffen und durch die starke Nutzung von elektronischen (Massen-) Kommunikationsmedien und die durchlässigen Grenzen jedoch weiter getrieben. So kann ein hohes Maß an Umweltirritation in das Projekt gelangen und die fluiden Strukturen ermöglichen eine schnelle Anpassung an sich verändernde Umwelterfordernisse.

Sind Open-Source-Software-Projekte der Prototyp der Organisation in der „next society“? Vermutlich nicht. Anderen Organisationen wird es kaum möglich sein, ihre Mitglieder nur auf der Grundlage intrinsischer Motivation zu rekrutieren. Auch wird die geringe Formalisierung in anderen Kontexten, in denen z.B. eine

Standardisierung der Produktionsabläufe erforderlich ist, in hohem Maße dysfunktional sein.
Trotzdem können Organisationen von Open-Source-Projekten lernen. Denn sie stellen ein sehr erfolgreiches Beispiel für die Nutzung des Internets zur Generierung von Innovation, zur Handlungskoordinierung und zum Wissensmanagement dar. Sie zeigen, wie man die Nutzer als Produzenten und Ideengeber einbinden kann und sie haben ein Modell von Führung etabliert, das kompatibel mit den fluiden Strukturen und komplexen Umweltbeziehungen ist.

Dadurch können Open-Source-Software-Projekte ein spannendes Feld weiterer organisationssoziologischer Forschung darstellen. Denn: Vieles, nach dem heutige Organisationen streben (flache Hierarchien, Dezentralisierung, Selbstorganisation, flexible Mitgliedschaftsmodelle, wertbasierte Integration über Unternehmenskulturen etc.) lässt sich an diesen Projekten studieren. Gleichzeitig zeigen sie einen Weg auf, wie sich Organisationen auf das neue Kommunikationsmedium Internet einstellen können und wie sich die Strukturen von Organisationen durch das Medium verändern.
Zudem bieten sich diese Projekte als Forschungsgegenstand an, da durch die Virtualität der Kommunikation und der damit verbundenen De-Lokalisierung aller Aktivitäten, ein Forschungsbereich freigelegt wird, welcher einer anspruchsvollen theoretischen Fundierung und Beschreibung bedarf.

Interview Transkript (Auszug)

MH: Du hast gerade den Maintainer angesprochen, welche Rolle spielt der für einzelne Projekte? Hat der strikte „Entscheidungsmacht“ - in Anführungszeichen - darüber, wie sich das Projekt weiterentwickelt, oder wird da versucht nen Konsens zu finden mit den anderen Entwicklern, die daran beteiligt sind?

I: Also schon immer Konsens, auf jeden Fall. Maintainer bedeutet bei uns nicht wirklich irgendwer mit Macht über irgendwelche Quelltext-Bereiche, sondern das sind Leute, die haben halt in nem gewissen Bereich nen Überblick, also im Normalfall irgendein Mensch, der sich mit irgendeinem Bereich auskennt ist um längen effektiver darin, irgendwelche Probleme zu beheben oder Erweiterungen durchzuführen. Also das geht halt gern mal um nen Faktor Zehn auseinander, wenn ich mich in nem Bereich nicht auskenne und will da an irgendwas arbeiten, brauch ich da vielleicht nen Tag für. Und der Mensch, der halt den Plan in dem Bereich hat, der halt nen Überblick darüber hat, braucht vielleicht nur ne Stunde dafür. Von daher ist es für das Gesamtprojekt relativ effektiv, wenn jeder Hauptsächlich in seinem Bereich arbeitet. Und es ist wiederum auch so, wenn irgendwer irgendwelche Erweiterungen vornimmt, die halt nicht irgendwelche anderen Sachen kaputt machen, werden sie halt im Normalfall aufgenommen. Und es ist nicht so, dass Maintainer dann sagen würden „ich mag irgendwas nicht-technisches an deinem Vorschlag nicht - du hast die falsche Hautfarbe oder was auch immer“ - blödes Beispiel - also das passiert halt nicht, dass aufgrund solcher Argumentationen irgendwelche Sachen nicht aufgenommen werden. Sondern das ist bei uns immer sachlich. Möglicherweise gibt es dann technische Probleme im ersten Versuch, den irgendwer implementiert hat. Dann erklärt der Mensch mit dem Plan dem Anderen halt „ja du kannst es so verbessern“ - im Normalfall machen die Leute das dann und dann kommt man am Ende zu nem Konsens, der für alle gut ist. Und das wird dann halt eingespielt.

…

MH: Was mich noch interessieren würde, ist: diese Mailinglisten, die ganzen Archive sind ja online Verfügbar und ich hab mir die mal nen bisschen angeguckt, das sind ja Berge von Emails die da pro Tag hin und her geschickt werden.

I: klar

MH: Liest man das alles als aktiver Entwickler?

I: Es ist verschieden. Also wie du sicher gesehn hast sind die Mailinglisten für verschiedene Bereiche relevant. Leute, die an Games arbeiten interessieren sich nicht zwangsläufig für KO-ce-Dinge oder so. Von daher haben wir da schon eine gewisse Aufteilung. Und

ansonsten ist es genauso, wie mit anderer Mail-Kommunikation auch, man sieht manche Sachen sind direkt an jemanden selbst adressiert, da antwortet man dann entsprechend drauf. Andere Sachen überfliegt man dann halt. Also man liest im Prinzip im Header worum es geht überfliegt die Eine oder Andere Mail und entscheidet für sich „okay, dieser Thread ist für mich gerade nicht interessant." Es gibt bei uns halt nen System, dass jeder SVN-Commit, also jede Änderung an unseren Quelltexten auslöst, dass ne Mail mit diesen Änderungen an ne Mailingliste verschickt wird. Da kommen halt wirklich ziemlich viele Mails bei zusammen. Und da gibt es schon einige Leute, die die wirklich alle lesen, um da nen Überblick drüber zu haben ob da gerade irgendwelche Leute irgendwelche dummen Dinge an wichtigen Bereichen tun oder so. Aber das sind maximal 5-10 Leute. Und ansonsten pickt man sich halt raus, welche Sachen für einen interessant sind und wieviel Zeit man für Mails lesen aufwenden will.

MH: Wenn diese 5-10 Leute, die du ansprachst, jetzt sehen, da hat einer totalen Murks gemacht, sind die dann auch berechtigt so eine Quellcode Zeile wieder zu entfernen?

I: Klar, natürlich. Das passiert aber eigentlich immer im Einverständnis mit irgendwelchen Leuten. Also mit dem Einverständnis des entsprechenden Menschen, der den Commit dann durchgeführt hat. Dem wird dann erzählt „war nicht so schlau, ändere das doch mal genau so und so" oder als Alternative gibt es dann „war nicht so schlau, ich hab das mal so und so geändert" oder „ich

habs für jetzt mal reverted - wieder entfernt - arbeite doch nochmal dran und ich habs halt entfernt, weil diese oder andere Dinge dadurch gerade brechen und wir können nicht so lange darauf warten, bis du das in ein, zwei Tagen fertig hast“. Das ist eigentlich immer nen normaler freundlicher Umgang miteinander, und es werden halt nicht diese Spiele: Irgendwer checkt was ein, jemand anderes reverted das gespielt. Also das kommt bei uns halt nicht vor.

Literatur

Albers, Tom (2010): Release Schedules. Online verfügbar unter http://www.omat.nl/2010/05/11/release-schedules/.

Baecker, Dirk (Hg.) (2007): Studien zur nächsten Gesellschaft. Frankfurt am Main: Suhrkamp.

Berger, Ulrike; Bernhard-Mehlich, Isolde (2006): Die Verhaltenswissenschaftliche Entscheidungstheorie. In: Alfred Kieser und Mark Ebers (Hg.): Organisationstheorien. 6. Aufl. Stuttgart: Kohlhammer, S. 169–214.

Blättel-Mink, Birgit; Hellmann, Kai-Uwe (Hg.) (2010): Prosumer Revisited. Zur Aktualität einer Debatte. Wiesbaden: VS Verlag für Sozialwissenschaften.

Bonazzi, Giuseppe (2008): Geschichte des organisatorischen Denkens. Hg. v. Veronika Tacke. Wiesbaden: VS Verlag für Sozialwissenschaften.

Brandt, Andreas (2009): Softwareentwicklung im Netzwerk. Kooperation, Hierarchie und Wettbewerb in einem Open Source-Projekt. München, Mering: Rainer Hampp.

Drucker, Peter F. (2001): The next society. In: The Economist, 03.11.2001.

Esposito, Elena; Baraldi, Claudio; Corsi, Giancarlo (1997): GLU: Glossar zu Niklas Luhmanns Theorie sozialer Systeme. Frankfurt am Main: Suhrkamp.

Ettrich, Matthias (1996): New Project: Kool Desktop Environment (KDE). Programmers wanted! Online verfügbar unter http://groups.google.com/group/de.comp.os.linux.misc/msg/cb4b2d67ffc3ffce?dmode=source&pli=1.

Fogel, Karl (2007): Producing Open Source Software: How to Run a Successful Free Software Project. Online verfügbar unter http://producingoss.com/.

Free Software Foundation: GPL. Online verfügbar unter http://www.fsf.org/licensing/licenses/gpl.html.

Hamel, Gary (2007): The future of management. Boston: Harvard Business School Press.

Hoffmann, Janette (1998): PKI "Let A Thousand Proposals Bloom" - Mailinglisten als Forschungsquelle. Online verfügbar unter http://duplox.wzb.eu/texte/gortex/index.html.

Holtgrewe, Ursula; Brand, Andreas (2005): KDE im Kontext: Open Source Software Entwicklung und öffentliche Güter. In: Manfred Moldaschl und Hajo Weber (Hg.): Wissen und Innovation. Beiträge zur Ökonomie der Wissensgesellschaft.

Horch, Heinz-Dieter (1985): Personalisierung und Ambivalent. Strukturbesonderheiten freiwilliger Vereinigungen. In: Kölner Zeitschrift für Soziologie und Sozialpsychologie 37, S. 257–274.

Jarvis, Stuart (2009): Repositioning the KDE Brand. Online verfügbar unter http://www.kdenews.org/2009/11/24/repositioning-kde-brand.

KDE.org (2009): Repositioning the KDE brand. Online verfügbar unter http://community.kde.org/Promo/Branding/Rebranding_KDE_v1.1.0.

KDE.org (2010a): Code of Conduct. Online verfügbar unter http://kde.org/code-of-conduct/, zuletzt aktualisiert am 2010.

KDE.org (2010b): Get a SVN Account. Online verfügbar unter http://techbase.kde.org/Contribute/Get_a_SVN_Account.

KDE.org (2010c): SVN Commit Policy. Online verfügbar unter http://techbase.kde.org/Policies/SVN_Commit_Policy, zuletzt aktualisiert am 2010.

Kieser, Alfred (2006): Der Situative Ansatz. In: Alfred Kieser und Mark Ebers (Hg.): Organisationstheorien. 6. Aufl. Stuttgart: Kohlhammer, S. 215–245.

Kieser, Alfred; Ebers, Mark (Hg.) (2006): Organisationstheorien. 6. Aufl. Stuttgart: Kohlhammer.

Knorr-Cetina, Karin (2005): Complex Global Microstructures: The New Terrorist Societys. In: Theory, Culture & Society 22, S. 231–234.

Knorr-Cetina, Karin; Brügger, Urs (2002): Globale Mikrostukturen der Weltgesellschaft: Die virtuellen Gesellschaften von Finanzmärkten. Online verfügbar unter www.prognosen-ueber-bewegung.de/files/124/file/knorr-cetina-mikrostrukturen.pdf.

Kügler, Sebasian (2008): The Start of Something Amazing with KDE 4.0 Release. Online verfügbar unter http://dot.kde.org/2008/01/11/start-something-amazing-kde-40-release.

Kühl, Stefan (1998): Wenn die Affen den Zoo regieren. Die Tücken der flachen Hierarchie. 5. Aufl. Frankfurt am Main/New York: Campus.

Landespressestelle Berlin (2009): Matthias Ettrich erhält Verdienstorden der Bundesrepublik Deutschland. Online verfügbar unter http://www.berlin.de/landespressestelle/archiv/2009/11/04/144882/.

Luhmann, Niklas (1975): Interaktion, Organisation und Gesellschaft. In: Niklas Luhmann (Hg.): Soziologische Aufklärung 2. Aufsätze zur Gesellschaftstheorie. Opladen: Westdeutscher Verlag, S. 9–20.

Luhmann, Niklas (1980): Gesellschaftsstruktur und Semantik. Studien zur Wissenssoziologie der modernen Gesellschaft. Frankfurt am Main: Suhrkamp (1).

Luhmann, Niklas (1987): Soziale Systeme. Grundriß einer allgemeinen Theorie. Frankfurt am Main: Suhrkamp.

Luhmann, Niklas (1988): Organisation. In: W. Küpper und G. Ortmann (Hg.): Mikropolitik. Opladen: Westdeutscher Verlag, S. 165–185.

Luhmann, Niklas (1999): Funktionen und Folgen formaler Organisation. 5. Aufl. Berlin: Dunker & Humblot.

Luhmann, Niklas (2006): Organisation und Entscheidung. 2. Aufl. Wiesbaden: VS Verlag für Sozialwissenschaften.

Luhmann, Niklas (2009): Die Realität der Massenmedien. 4. Aufl. Wiesbaden: VS Verlag für Sozialwissenschaften.

Mintzberg, Henry (1998): The Innovative Organization. In: Henry Mintzberg und James Brian Quinn (Hg.): The Strategy Process. Concepts, Contexts, Cases. 3. Aufl. New Jersey: Prentice Hall International, S. 679–692.

Mintzberg, Henry; Quinn, James Brian (Hg.) (1998): The Strategy Process. Concepts, Contexts, Cases. 3. Aufl. New Jersey: Prentice Hall International.

pcgames.de (2010): Google Android: Googles Handy-Betriebssystem überholt Apple iPhone OS. Online verfügbar unter http://www.pcgames.de/Google-Firma-97880/News/Google-Android-Googles-Handy-Betriebssystem-ueberholt-Apple-iPhone-OS-747579/.

Picot, Anrnold; Fiedler, Marina (2008): Organisation von Innovation. Koordination und Motivation von Open-Source-Software-Projekten. In: Sonja Sackmann (Hg.): Mensch und Ökonomie. Wie sich Unternehmen das Innovationspotenzial dieses Wertespagats erschließen. Wiesbaden: Gabler.

Raymond, Eric S. (2001): How To Become A Hacker. Online verfügbar unter http://www.catb.org/esr/faqs/hacker-howto.html#believe2.

Stallman, Richard (1985): The GNU Manifesto. Online verfügbar unter http://www.gnu.org/gnu/manifesto.html.

Stallman, Richard (1992): Why Software Should Be Free. Online verfügbar unter http://www.gnu.org/philosophy/shouldbefree.html.

Stallman, Richard (1998): Das GNU Projekt. Online verfügbar unter http://www.gnu.org/gnu/thegnuproject.de.html.

Stichweh, Rudolf (2005): Inklusion und Exklusion. Studien zur Gesellschaftstheorie. Bielefeld: transkript.

Taubert, Nils (2006): Produktive Anarchie? Netzwerke freier Softwareentwicklung. Bielefeld: Transcript.

Toffler, Alvin (1980): The third wave. New York: Morrow.

Volkmann, Ute (2010): Sekundäre Leistungsrolle. Eine differenzierungstheoretische Einordnung des Prosumenten am Beispiel des "Leser-Reporters". In: Birgit Blättel-Mink und Kai-Uwe Hellmann (Hg.): Prosumer Revisited. Zur Aktualität einer Debatte. Wiesbaden: VS Verlag für Sozialwissenschaften, S. 206–220.

Weick, Karl E. (1996): Drop your tools: An allegory for organizational studies. In: Adminstrative Science Quarterly, S. 301–313.

Zimmermann, Thomas (2004): Open Source und Freie Software. Soziale Bewegung im virtuellen Raum? In: Robert A. Gehring und Bernd Lutterbeck (Hg.): Open Source Jahrbuch 2004. Zwischen Softwareentwicklung und Gesellschaftsmodell. Berlin: Lehmanns Media.

Zündorf, Lutz (1986): Macht, Einfluß, Vertrauen und Verständigung. Zum Problem der Handlungskoordinierung in Arbeitsorganisationen. In: Rüdiger Seltz, Ulrich Mill und Eckardt Hildebrandt (Hg.): Organisation als soziales System. Kontrolle und Kommunikationstechnologie in Arbeitsorganisationen. Berlin: Edition Sigma, S. 33–56.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und dabei auf keine anderen als die im Literaturverzeichnis aufgeführten Quellen und Hilfsmittel zurückgegriffen habe. Ferner sind alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht.

Jens Martin Heuer
Bielefeld, 27. Juli 2011