7. Wissenstransfer in betriebsübergreifenden Innovationsprozessen durch Open Source Communities

Patrick Feuerstein und Heidemarie Hanekop

7.1 Einleitung

In der Softwarebranche hat sich seit den 1990er Jahren mit der Open Source Softwareentwicklung (OSS) eine besondere Form gemeinschaftlicher Softwareentwicklung etabliert, die in einigen Feldern auch für Unternehmen mittlerweile ökonomisch höchst relevant ist (O'Mahony & Lakhani 2011; Schrape 2015). Es sind – neben einer Vielzahl von kleineren OSS-Projekten – einige große Open Source Softwareprodukte entstanden, die sich zu einer nachhaltig konkurrenzfähigen Alternative zu marktförmigen Softwareprodukten großer IT-Unternehmen entwickelt haben. Sehr bekannt ist z.B. das Betriebssystem Linux, das sich als Alternative zu Microsofts lange Zeit dominierendem Windows-Betriebssystem herausgebildet hat. Andere Beispiele sind Apache als Web-Server, Firefox als Webbrowser, Java als Programmiersprache, das Content-Managementsystem Typo3 oder als aktuelles Beispiel das Cloud-Management-System OpenStack.

Als besonderer Vorteil wird OSS-Communities zugerechnet, durch die offene und gemeinschaftlich organisierte Form der Wissensproduktion eine Struktur für den Wissensaustausch zwischen vielen, wechselnden, heterogenen Akteuren bereit zu stellen, die zunehmend auch über die Grenzen reiner OSS-Communities hinweg strategisch eingesetzt bzw. adaptiert wird: Auch Unternehmen sichern sich zunehmend auf diese Weise den Zugang zu externem Wissen und nutzen an OSS angelehnte Ansätze im Rahmen kollaborativer Innovationsprozesse (siehe u.a. Dahlander

& Magnusson 2005; West & Lakhani 2008; O'Mahony 2007; Von Hippel 2005; Mateos-Garcia & Steinmueller 2008; Giuri et al. 2008; David & Shapiro 2008; West & Gallagher 2006; West & O'Mahony 2008). Bei der großen Beachtung, die solche Formen der gemeinschaftlichen Wissenserzeugung erfahren haben, ist es überraschend, dass bisher nur wenige Studien untersucht haben, welche Mechanismen des Wissenstransfers und damit einhergehend auch welche spezifischen Problemlagen sich für die beteiligten Unternehmen bei solchen Formen community-basierter, kollaborativer Innovationsprozesse ergeben (für Ausnahmen, siehe West & O'Mahony 2008; Dobusch & Quack 2011).

Der folgende Beitrag versucht, genau dieser Frage nachzugehen: Unsere Ausgangsthese ist, dass OSS ein Modell für erweiterte Formen des Wissenstransfers über Organisationsgrenzen hinweg darstellt. Unsere These ist weiter, dass OSS-Communities durch ihre Eigenschaften besonders geeignet sind, Mechanismen zum Teilen von überwiegend impliziten oder „taken-for-granted" Wissensbestandteilen bereitstellen, die in den jeweils individuellen Erfahrungen und beruflichen Hintergründen der beteiligten Entwickler verankert sind. Wir behaupten, dass OSS-Communities soziale Praxen für die grenzüberschreitende Explikation solcher Wissensbestandteile schaffen, die auch implizites Wissen zugänglich machen. Die spezifische soziale Ordnung von OSS-Communities, mit den hierfür etablierten Institutionen, Regeln und auch sozialen Praxen, unterstützt – anders als z.B. marktförmige Formen der Governance – den Wissenstransfer über Unternehmensgrenzen hinweg. Austausch- und Lernprozesse, die bei marktförmigen Governance-Formen meist bilateral und im Einzelfall gezielt aufgebaut werden müssen, sind in OSS-Communities fest institutionalisiert. Die Implikationen für die Unternehmen, die sich dieser Möglichkeit des Zugangs zu externem Wissen bedienen, so unser Argument weiter, sind dabei jedoch weitreichend und greifen tief in die internen, betrieblichen Unternehmensprozesse ein. So ist der zweite Teil des hier entwickelten Arguments, dass Community-basierte Innovationsprozesse für die beteiligten Unternehmen mit systematischen Steuerungsproblemen einhergehen, die von diesen organisatorisch und personell bewältigt werden müssen.

Diese Argumente sollen im Folgenden anhand der Fallstudie eines OSS-Projektes (für die genaue Beschreibung der Datengrundlage, siehe Kapitel 2) belegt werden. Zunächst werden im ersten Schritt die Probleme des Wissenstransfers in überbetrieblichen Innovationsprozessen skizziert. Anschließend werden OSS-Communities als Sonderformen gemeinschaftlicher Governance in ihren wesentlichen Eigenschaften skizziert und zentrale Institutionen und Strukturmerkmale herausgestellt. Im vierten Abschnitt schließlich wird anhand der Fallstudie einer OSS-Community dokumentiert, auf welche Weise die besonderen Eigenschaften der OSS-Community zu einem gelingenden Wissenstransfer zwischen den beteiligten Akteuren beitragen. Anschließend fokussieren wir auf die dabei für die Unternehmen entstehenden Steuerungsprobleme, bevor wir schließlich die von den betrieblichen Akteuren entwickelten Lösungsansätze beschreiben. Ein Fazit schließt das Kapitel ab.

7.2 Das Problem des Wissenstransfers in verteilten Innovationsprozessen

Innovationsprozesse, die über Unternehmensgrenzen hinausgehen, potenzieren ein Problem, das die Innovationsforschung auch schon für innerbetriebliche Innovationsprozesse herausgearbeitet hat: Das für Innovationen notwendige und gewünschte Wissen ist gewöhnlich über verschiedene Wissensträger verstreut. Um das für Innovationen nötige Wissen zu erzeugen, müssen also verschiedene Akteure zusammengebracht werden, mit allen Folgen, die aus den Problemen kollektiven Verhaltens in und zwischen ausdifferenzierten und vermachteten Organisationen in Bezug auf individuelle Interessenlagen aber auch aus den unterschiedlichen Entstehungskontexten von Wissen folgen können.

So ist einerseits das für Innovationen relevante Wissen in der Regel nicht unabhängig von den beteiligten Akteuren zu stabilisieren. Wissen und auch Wissensbestände können nicht als Menge allgemeingültiger, wahrer Aussagen über die Welt begriffen werden. Vielmehr muss Wissen, wie Heidenreich (2003) ausführt, als „lernbereite" Deutungsschemata verstanden werden, die den natürlichen und sozialen Lebensbedingungen der Menschen einen Sinn geben und die ihr praktisches Verhalten regeln. Nach dieser Definition ist Wissen damit zum einen ständig vorläufig (und damit „lernbereit"), zum anderen aber auch situationsabhängig und damit seinem Wesen nach höchst kontextabhängig, da es als soziale Konstruktion aus unterschiedlichen menschlichen Lebensbedingungen heraus erwächst (siehe z.B. Berger & Luckmann 1984; siehe auch die Ausführungen der Einleitung in diesem Band). Für die Untersuchung betriebsübergreifender Innovationsprozesse ist ein solcher Wissensbegriff nützlich, da er die für Unternehmen komplizierte Aufgabe der Integration verteilter Wissensbestände fassen kann. Das für Innovationen benötigte Wissen ist über mehrere Akteure an verschiedenen Orten, z.B. in verschiedenen Organisationen, verteilt und an den jeweiligen Entstehungskontext gebunden. Die Kontextgebundenheit des Wissens macht sich vor allem daran fest, dass Wissen nicht beliebig expliziert und kommuniziert werden kann, eine Tatsache, auf die Polanyi (1985) in seiner klassischen Unterscheidung zwischen implizitem und explizitem Wissen hingewiesen hat. Nach Polanyi ist nur ein kleiner Teil des Wissens eindeutig explizierbar und kommunizierbar, da Wissen stets auf nicht hinterfragten Annahmen der Akteure über die Wirklichkeit, die in ihrer jeweiligen (in diesem Fall betrieblichen) Lebenspraxis eingebettet sind, beruhen. Nach Polanyi (1985) ist es gerade dieser (über-)individuelle Background der Akteure, der das Verstehen und sinnvolle Anwenden expliziter, kommunizier- und formalisierbarer Wissensarten, wie z.B. technischer Artefakte und Routinen, überhaupt erst möglich macht (vgl. auch Tsoukas 1996; Heidenreich 1995).

Das zweite Problem betriebsübergreifender Innovationsprozesse kann nach Krogh (2011) als Problem kollektiven Verhaltens rekonstruiert werden: Wie können Akteure aus ganz verschiedenen Organisationen, mit ganz unterschiedlichen Interes-

sen, dazu gebracht werden, ihr jeweiliges Wissen in gemeinsamen Innovationsprozessen zu teilen und gemeinsam zu kombinieren? Diese Perspektive betont vor allem zwei relevante Dimensionen des Problems betriebsübergreifender Innovationsprozesse. Die erste Dimension weist darauf hin, dass die Kombination unterschiedlicher Wissensbestände als soziale Praxis verstanden werden muss: Wissen kann demnach nicht einfach durch formale Prozeduren und Kommunikationen übermittelt werden, sondern erfolgreiche Vermittlung setzt vielmehr auch den Transfer des impliziten Wissens, das Teilen des jeweiligen Backgrounds der Akteure in einer gemeinsamen (und gemeinsam zu entwickelnden) sozialen Praxis voraus (Krogh 2011). Dieser Auffassung folgend, wird Wissen in verteilten Innovationsprozessen weniger *transferiert* als in sozialen Interaktionen stets *neu geschaffen*. Welche Anreize für die einzelnen Akteure jeweils gegeben sein müssen, damit sie sich in solchen gemeinsamen Austauschprozessen beteiligen und ihr Wissen einbringen, ist daher auch die zweite Dimension des Problems betriebsübergreifender Innovationsprozesse, auf die dieser Ansatz hinweist: Können Unternehmen bei internen Innovationsprozessen zumindest noch auf gemeinsame organisatorische Sozialisationsprozesse und eine zumindest z.T. gemeinsame Zielsetzung der Akteure im Rahmen gemeinsamer Ziele der Organisation vertrauen, so müssen in unternehmensübergreifenden Innovationsprozessen Akteure zusammengebracht werden, die in ganz unterschiedlichen organisatorischen Kontexten situiert sind und ganz unterschiedliche Zielsetzungen verfolgen können. Das Problem des Teilens impliziter Wissensbestandteile im Rahmen sozialer Praxis ist in dieser Konstellation noch einmal verschärft, da die Akteure weder an gemeinsame Erfahrungen und Kontexte anknüpfen, noch auf die im Rahmen der Organisation gesetzten gemeinsamen Zielsetzungen vertrauen können. Unternehmen in unternehmensübergreifenden kollaborativen Innovationsprozessen stehen daher vor der Herausforderung, solche Formen der Organisation und Praxis zu finden, die es ermöglichen, nicht nur explizite, sondern auch implizite Wissensbestandteile zu teilen.

7.3 Gemeinschaftliche Governance verteilter Innovationsprozesse durch Open Source Communities

Betriebsübergreifende Innovationsprozesse können durch Unternehmen in unterschiedlicher Weise organisiert und koordiniert werden. Unter Rückgriff auf Hollingsworth & Boyer (1997) können diese unterschiedlichen Koordinationsweisen sozialen Handelns von Akteuren als unterschiedliche Governancemechanismen gefasst werden (vgl. auch Windeler 2012). In diesem Beitrag untersuchen wir die Möglichkeiten und Probleme, die für Unternehmen entstehen, die versuchen, mithilfe gemeinschaftlicher Governance – durch Teilnahme an OSS-Communities – den Zugang zu externem Wissen herzustellen und zu sichern.

Ungeachtet der herrschenden Vielfalt der Typologisierungsversuche hat sich Gemeinschaft neben Markt, Hierarchie (Organisation) und Netzwerk als ein eigenständiger Governancemechanismus in der Debatte etabliert (Streeck & Schmitter 1985; Hollingsworth 2000; Gläser 2007). Gemeinschaft wird in der Governanceforschung dabei als ein Koordinationsmechanismus begriffen, der auf der Handlungssteuerung der Mitglieder durch ein Gefühl der Zugehörigkeit beruht und sich durch solidarisches Verhalten in Bezug auf die Ziele der Gemeinschaft auszeichnet. Die Zugehörigkeit muss dabei nicht auf festen Regeln und Festlegungen beruhen, sondern kann auch durch die subjektive Wahrnehmung der Akteure hergestellt werden (vgl. Gläser 2007; Streeck & Schmitter 1985). Wichtig ist, dass mit der Zugehörigkeit die Identität der Mitglieder betroffen ist:

> *„Man agiert, weil man anderen in einem spezifischen Merkmal gleicht, und man agiert deswegen auf eine spezifische Weise."* (Gläser 2007)

Gemeinschaften können diese kollektive Identität auf unterschiedlichen Grundlagen entwickeln, Gemeinschaften auf Grundlage gemeinsamer Aktivitäten (z.B. Fanprojekte, Sportgemeinschaften) finden sich hier genauso wie professionelle oder berufliche Gemeinschaften (z.B. die breit diskutierten „communities of practice", siehe Wenger 2000), die gemeinsame professionelle Standards und Vorgehensweisen entwickeln und sich zudem häufig formal in berufsständischen Organisationen zusammenfinden und gemeinsame Interessen verfolgen.

Ist mit der kollektiven Identität der Mitglieder der zentrale Koordinationsmechanismus gemeinschaftlicher Governance – und damit das zentrale Spezifikum gerade auch in Abgrenzung zu Markt, Netzwerk und Organisation – benannt (vgl. Streeck & Schmitter 1985), so reicht diese Eigenschaft zur Charakterisierung der Koordination kollektiven Handelns durch Gemeinschaften nicht aus.

So bestimmen Dolata & Schrape (2014) Gemeinschaften – über die bereits erwähnte kollektive Identität hinaus – über zwei weitere Merkmale. Zum einen verweisen sie auf „Institutionalisierungsdynamiken, die kollektives Handeln auf der Basis eigener, vornehmlich informeller Regeln, Normen und Organisierungsmuster ermöglichen, strukturieren und stabilisieren", und zum anderen auf „interne Differenzierungsprozesse, in denen sich mit der Zeit organisierende Kerne und meinungsführende Aktivisten mit umliegenden Peripherien aus unterstützenden Teilnehmern herauskristallisieren" (S. 19, für eine ähnliche Definition, siehe auch Dobusch & Quack 2011).

Es ist gerade diese „Institutionalisierung des Kollektiven" (Dolata & Schrape 2014), die Gemeinschaften auch zu strategisch handlungsfähigen Akteuren macht. Die gemeinsam geteilten Normen und informellen Regeln stellen die Grundlage für kollektives Handeln dar und ermöglichen Gemeinschaften damit eine gewisse Eigenständigkeit und Eigengesetzlichkeit gegenüber externen Akteuren. So bilden kollektive Institutionen auch einen Schutz vor der einseitigen Vereinnahmung durch

Unternehmensinteressen. Gleichzeitig sind es – wie später gezeigt wird – gerade diese institutionellen Grundlagen und Verkehrsformen von Gemeinschaften, die für Unternehmen zu Steuerungsproblemen in verteilten Innovationsprozessen führen.

Mit der Berücksichtigung interner Differenzierung und der Etablierung von Gemeinschafts*organisationen* als Eigenschaft von Gemeinschaften mischen sich in gewisser Weise organisatorische Elemente mit gemeinschaftlicher Governance (Gläser 2007; Wiesenthal 2005). Es ist daher wichtig, die besondere Qualität von Gemeinschaftsorganisationen im Gegensatz zu formalen Organisationen herauszustellen. Als wesentliche Unterschiede heben Dobusch & Quack (2011) in dieser Hinsicht zum einen die Regelung der Zugehörigkeit hervor, die in Gemeinschaften häufig auf Beiträgen zu dem gemeinschaftlichen Zielen und nicht auf formalen Zugangsberechtigungen und -bedingungen beruht. Eng damit zusammen hängt auch die Frage des Einflusses innerhalb der Gemeinschaft und des Modus der Entscheidungsfindung. Gemeinschaftliche Entscheidungen werden häufig konsensorientiert und nach explizit meritokratischen und nicht nach formal rechtlichen Regelungen gewährt.

Gemeinschaften als strategisch handlungsfähige Akteure lassen sich also generell mit den drei Merkmalen einer *internen Institutionalisierung*, der Herausbildung einer eigenen *kollektiven Identität* und *interner Differenzierungs- und Organisationprozesse* bestimmen (Dolata & Schrape 2014; für ähnliche Definitionen siehe z.B. Gläser 2007; Dobusch & Quack 2011).

Die in diesem Beitrag im Fokus stehenden OSS-Communities können als eine besondere Form gemeinschaftlicher Governance verstanden werden (West & Lakhani 2008), die die bereits diskutierten Merkmale gemeinschaftlicher Governance teilen. Jedoch weisen OSS-Communities auch Eigenschaften auf, die über die obige allgemeine Bestimmung gemeinschaftlicher Governance hinausgehen. Open Source Communities sind Produktionsgemeinschaften (Gläser 2006). Grundlegend für die kollektive Identität der OSS-Community ist das gemeinsame Ziel der Erstellung und Verbreitung einer bestimmten Open Source Software. Diese Software entsteht aus selbstbestimmten Beiträgen von freiwilligen Entwicklern. Die über Beiträge zur Produktentwicklung gestiftete Mitgliedschaft in der Community sorgt dafür, dass nicht mit Weisungen und Pflichten bezogen auf die Beiträge Einzelner agiert werden kann. Vielmehr zeichnen sich OSS-Communities durch reziproke Strukturen der gegenseitigen Unterstützung und des solidarischen Umgangs miteinander aus. Eine häufig nicht explizit formulierte Etikette beinhaltet Mindestanforderungen des gemeinsamen Miteinanders und stellt die institutionalisierte Grundlage dafür dar, dass gemeinsames Handeln möglich wird, Vertrauen entsteht und Leistungen für das gemeinsame Ziel anerkannt bzw. nicht-solidarisches Verhalten sanktioniert wird. Gerade auf dieser normativen Ebene bestehen starke Analogien zu herkömmlichen Gemeinschaften: Sie sind solidarisch, kooperativ und wesentlich durch gemeinsame Interessen und Ziele verbunden. Open Source Communities werden in der Literatur als eine modernisierte Variante von Gemeinschaft diskutiert, auf konkrete Ziele ausgerichtet, leistungsorientiert und meritokratisch, frei von persönlichen

Abhängigkeiten und Zwang, dabei offen im Zugang und einfachem Exit (z.B. O'Mahony & Lakhani 2011). Trotz der damit verbundenen Volatilität bilden OSS-Communities im kollektiven Entwicklungsprozess eine soziale Struktur heraus. Die besondere technische Infrastruktur ist diesem Typ von Gemeinschaftsorganisation charakteristisch, wie Dolata & Schrape (2014) herausstellen. OSS-Communities organisieren sich im Rahmen technischer Infrastrukturen, die sowohl die Formen der technischen als auch der sozialen Interaktion entscheidend vorstrukturieren (Dolata & Schrape 2014). So ist der technische Prozess der Open Source Software Entwicklung weitgehend transparent, er wird über kollaborative Entwicklungsplattformen im Web (z.B. GitHub) organisiert, auf denen sich Entwickler und Anwender aus aller Welt beteiligen können, um die neuen Funktionen in ihrer Systemumgebung zu testen und/oder Änderungen vorzuschlagen[1]. Spezielle Lizenzen stellen sicher, dass der gemeinsam generierte Code nicht proprietär verwendet werden darf und offen bleibt (für einen Überblick über häufige Lizenzen, siehe z.B. http://opensource.org/licenses). Über die Speicherung des Codes hinaus bieten viele Plattformen begleitende technisch basierte Informations- und Kommunikationskanäle (Mailinglisten, IRC, u.ä.), die das gemeinsame Arbeiten an geteilter Code-Basis erleichtern. Eine wichtige Funktion dabei ist häufig auch die Speicherung der vorherigen Kommunikation in Archiven, die einmal gefällte Entscheidungen und Auseinandersetzungen für später hinzugekommene Entwickler nachvollziehbar macht. Die Entwicklungsplattformen können damit als das zentrale Element für das Teilen expliziten oder explizierten Wissens in OSS-Communities angesehen werden.

Die Open Source Bewegung ist in den 1990er Jahren als Alternative zur kommerziellen Softwareproduktion marktbeherrschender Unternehmen entstanden. Mit der Ausbreitung und dem Erfolg der Open Source Projekte und dem zunehmende Interesse und Engagement von Unternehmen tritt die Konfrontation in den Hintergrund und es entstehen Verbindungen zwischen OSS-Projekten und kommerziellen Aktivitäten, die nicht ohne Wirkung auf die interne Governance einiger Communities bleiben. Trotz dieser Ausdifferenzierung wird zumindest für die als „community-managed" bezeichneten Open Source Projekte (West & O'Mahony 2005) angenommen, dass die interne Governance durch die beschriebenen Merkmale gemeinschaftlicher Governance geprägt ist.

Wir werden in der nun folgenden Fallanalyse genauer untersuchen, inwiefern diese Eigenschaften von OSS-Communities den Wissenstransfer zwischen den unterschiedlichen beteiligten Wissensträgern ermöglichen und welche spezifischen Steuerungsprobleme sich für die beteiligten Unternehmen daraus ergeben.

[1] In der Praxis unterscheiden sich OSS-Communities dahingehend, wie offen die Codebasis für Uploads von unbekannten Entwicklern und Anwendern ist. Häufig gibt es organisationale Reglungen, mit denen der schreibende Zugriff auf die Codebasis beschränkt wird. Nichtsdestotrotz steht es allen offen, eigene „forks", also Abspaltungen des Programmcodes, vorzunehmen.

7.4 Fallstudie: Wissenstransfer, Steuerungsprobleme und -lösungen im OSS-Projekt „MSB“

7.4.1 Zur Entwicklung und Struktur der OSS-Community „MSB“

Die Gründer des von uns untersuchten Open Source Projektes beginnen Anfang der 1990er Jahre eine Software zu entwickeln, die die Interoperabilität zwischen Windows und Unix- bzw. Linux-basierten Client-Server Netzwerken ermöglicht. Microsoft hat zu diesem Zeitpunkt eine Monopolstellung bei den PC-Betriebssystemen und beginnt diese auf den bis dato von anderen Anbietern beherrschten Markt von Client-Server Netzwerken auszudehnen. Die von Microsoft auf dieser Grundlage etablierte Netzwerk-Software basiert zwar auf einem ursprünglich offenen Protokoll, das aber von Microsoft zu einem geschlossenen Netzwerk-System weiterentwickelt wird, dessen Spezifikationen nicht offengelegt und dokumentiert werden. Windows-PCs werden durch diese Netzwerk-Software in die Lage versetzt, mit Windows-Servern und anderen Clients zu kommunizieren, z.B. um gemeinsam Drucker oder Speicher zu nutzen. Die bis dato vorherrschenden Server-Systeme können nicht mit Windows-PCs kommunizieren, d.h. sie können den mittlerweile als Clients vorherrschenden Windows-PCs keine Druck und Speicherdienste anbieten. Die Anbieter dieser Server- und Netzwerk-Technologien, darunter die bisherigen Marktführer wie Novell und Hardware-Hersteller wie IBM, SUN und HP, geraten hierdurch unter Druck und verlieren rasant an Bedeutung in dem sich dynamisch verändernden Netzwerksegment. Allerdings sind deren Systeme selbst proprietär, heterogen und nicht wirklich interoperabel (obwohl meist Unix-Varianten). Als Konkurrenten auf dem Markt schaffen sie es nicht, eine wirkungsvolle, gemeinsame Strategie gegenüber Microsoft zu entfalten. Ihre Versuche, den Vormarsch von Microsoft im Segment der Netzwerksysteme juristisch zu stoppen, verfehlen (zunächst) ihr Ziel. Als wirkungsvoller hingegen erweist sich die Strategie der Hauptentwickler und Gründer der OSS-Community, die beginnen eine Software zu entwickeln, die die Kommunikation von Unix und Linux Rechnern in Windows Netzwerken ermöglicht. Dies betrifft zunächst die Einbindung Unix-basierter Clients in bestehende Windows-Netzwerke, später aber auch die Möglichkeit, Unix-basierte Server in Windows-Netzwerken zu betreiben und Dienste für Windows-Clients bereitzustellen.

Die Community kommt aus der in den 1990er Jahren gerade erstarkenden Linux-Bewegung, in der viele Entwickler gemeinsam alternative Software entwickeln, die frei und offen verbreitet und stetig verbessert wird. Eine starke Triebfeder der Bewegung richtete sich insbesondere gegen die Monopolstellung von Microsoft.

Die Verantwortung und auch das Copyright (als Open Source) für die Software liegt bei der Community, sie organisiert den Produktionsprozess nach den Regeln der Open Source Lizenz Gnu Public Licence (GPL) und im Rahmen einer Open Source Community.

In der ersten Phase bestand die wesentliche Aufgabe der Community darin, Microsofts proprietäre Protokolle durch aufwendige Testverfahren und viel Improvisation aufzudecken und in eine neue Software zu überführen. Die besondere Herausforderung bestand darin, dass die Entwickler zunächst herausfinden mussten, wie die Windows-Rechner miteinander kommunizieren, bevor diese Kommunikation in der neuen Linux-Software simuliert werden kann. Dieser erste Schritt war eine komplexe, experimentelle Forschungsaufgabe, die dadurch erschwert wurde, dass Microsoft seine Netzwerk-Protokolle selbst dynamisch weiterentwickelte. Das Aufdecken des Kommunikationsverhaltens der proprietären Windows-Netzwerksoftware und dessen Nachbau als Linux-Programm wurde zu einer Daueraufgabe der Entwickler in der Community, deren Zahl sich mit der Funktionalität und Wirkung, die das „MSB" Produkt erzielt, Anfang der 2000er Jahre rasch vergrößerte.

Wegen der strategischen Bedeutung dieser Software als Interoperabilitätsschnittstelle zu Windows-Netzwerken ist die „MSB" Community für Konkurrenten von Microsoft in diesem Feld höchst interessant. Denn die Community entwickelt jenseits von singulären Unternehmensinteressen ein Produkt, das geeignet ist, der Dominanz von Microsoft in diesem Bereich entgegenzutreten, was keinem der großen Unternehmen gelungen war. In der Folge nimmt die Zahl der in einem Konkurrenzunternehmen von Microsoft tätigen Entwickler in der Community stetig zu. Allerdings ist es keineswegs so, dass die Community von Unternehmen mit dem Zweck gegründet wurde, diese Software zu produzieren. Unter Berücksichtigung der Differenzierung von West & O'Mahony (2005) kann die hier untersuchte Community also als „firm-sponsored", aber trotzdem „community-founded" charakterisiert werden. Die Initiative zur Entwicklung der Software ging auf einen Entwickler zurück, der in der Open Source Bewegung bereits einen Namen als genialer Entwickler und Forscher hatte. Ihm schlossen sich rasch einige weitere Entwickler an, so dass es gelang, ab Mitte der 1990er Jahre eine funktionsfähige Software mit grundlegenden Funktionen zu veröffentlichen. Seit den 2000er Jahren wird die Entwicklung der Software zu einem wesentlichen Teil von Entwicklern aus Unternehmen geleistet, die dies hauptsächlich im Rahmen ihrer Arbeitszeit und im Auftrag des Unternehmens tun. Diese Konstellation ist heute in vielen großen OSS-Projekten verbreitet (vgl. West & O'Mahony 2005; Schrape 2015).

Ein entscheidender Einschnitt erfolgt 2007 als der Europäische Gerichtshof u.a. auf Druck der an der Community beteiligten Akteure in einem Grundsatzurteil Microsoft zwingt, im Sinne der Interoperabilität mit anderen Systemen und Anbietern, die Spezifikationen seines Netzwerkprotokolls zu dokumentieren und öffentlich zugänglich zu machen. Dieser Einschnitt markiert eine Wende in Microsofts bis dahin verfolgter Strategie gegenüber der MSB-Community. War Microsoft vorher bemüht, die Arbeit der Community zu erschweren und sah diese in erster Linie als Gefahr für den eigenen Unternehmenserfolg an, so zeigt sich Microsoft nun bereit, über die vom Gericht geforderten Maßnahmen hinaus, in verschiedenen Formen (siehe unten) mit der Community zu kooperieren. Das neue strategische Leitbild der *Interoperabilität* führt dazu, dass Microsoft die Community aktiv dabei unterstützt,

seine jeweils neuesten Entwicklungen im Bereich von Server-Workstation-Netzwerken in der „MSB"-Software zu implementieren (für eine Auseinandersetzung mit Microsofts Strategiewechsel, siehe Lange 2009). Microsoft ist nicht selbst mit eigenen Entwicklern in der Community aktiv, sucht aber eine enge Kooperation über gemeinsame Veranstaltungen und Meetings und fördert die Community. Für Microsoft ist die Community nun ein strategisch wichtiger Partner, der gewährleistet, dass die neuen Entwicklungen von Microsoft (wie z.B. die neue Version des Netzwerkprotokolls SMB 3) nicht nur in der Windows-Welt zum Einsatz kommen, sondern auch unter anderen Plattformen anschlussfähig sind (z.B. in großen Storage Lösungen oder in der „Cloud" verschiedener Anbieter). Letztlich versucht Microsoft damit, sich über einen anderen Weg die Vorherrschaft seiner eigenen Protokolle und Lösungen zu sichern.

Kennzeichnend für die *Mitgliederstruktur* der MSB-Community ist einerseits ihre Konstanz, andererseits ihre enge *Verknüpfung mit Unternehmen*, deren Strategien in unterschiedlicher Weise mit dem von der Community verbreiteten Softwareprodukt verknüpft sind. Die Mitgliedschaft in der Community ist nicht für alle offen. Zwar dürfen alle den entstehenden Code lesen und nutzen, bevor allerdings Personen das Recht erhalten, selbständig Beiträge zur Codebasis zu leisten, müssen Sie in das Team aufgenommen werden. Wie für OSS-Communities üblich (s.o.) wird man Mitglied durch aktive Beiträge zur Entwicklung der Software. Prinzipiell kann jeder einen Beitrag vorschlagen, ob dieser aber dann auch in die offizielle Community-Software aufgenommen wird, darüber entscheiden die Mitglieder in einem Code-Review. Vor der Aufnahme als Mitglied steht oft ein durchaus aufwendiger Lern- und Prüfungsprozess, während dessen der Interessent sich in die Programmstruktur einarbeiten und seine Kompetenz zur Mitarbeit durch sinnvolle und qualitativ hochwertige Beiträge unter Beweis stellen muss. Mit den Beiträgen zeigt der Interessent über die reinen Programmierkenntnisse hinaus jedoch auch, dass er die teamförmige Arbeitsweise der Community beherrscht und sich an die informellen Regeln der Community hält, wie ein Entwickler hervorhebt:

> *„Das ist so ein Initiations-Ritus. […] Die „MSB"-Teammitgliedschaft definiert sich im Prinzip über die Schreibrechte auf dem öffentlichen „MSB"-Quellcode-Repository. […] Und man bekommt sie, indem man sich eine Weile dort getummelt hat, gezeigt hat – da immer mal wieder mitarbeitet […] Wenn man [das – PF]]in genügendem Umfang und Kontinuität getan hat, dann wird in der Regel jemand daher gehen und sagen, guck mal, diese Person hat doch schon so schön – wollen wir die nicht mal ins Team aufnehmen? Dann stimmt das Team intern ab auf einer Team-internen Mailingliste. Man wird dort schön willkommen geheißen, wenn man sich gut anstellt und nett kommuniziert. Also wenn man sich nicht stoffelig anstellt, dann wird man mit offenen Armen empfangen."* (FS17-IT20-3)

Fast alle Mitglieder der Community sind professionelle Entwickler oder Informatiker aus Unternehmen. Die Zahl der zu einem bestimmten Zeitpunkt aktiven Hauptentwickler schwankt in einer Größenordnung zwischen 25 bis 40. Einige sind seit

mehr als 15 Jahren oder sogar seit der Anfangszeit kontinuierlich dabei, andere sind für einige Monate oder Jahre aktiv und wenden sich dann wieder stärker anderen Aufgaben zu. Insgesamt waren etwa 100 Entwickler maßgeblich an der Entwicklung beteiligt. Der Initiator der Community hat sich in den letzten Jahren zurückgezogen, alle anderen Gründer sind weiterhin aktiv. Gleichzeitig kommen stetig einige wenige neue Entwickler hinzu, die oft rasch zu neuen Hauptentwicklern werden. Die Gesamtzahl der Entwickler, die seit 2008 beigetragen haben, liegt bei 360, viele tragen einmal wenige Beiträge bei, wenn sie an einem bestimmten Problem arbeiten, das sie beruflich beschäftigt.

Nicht ganz selten wechseln Entwickler während ihrer Mitarbeit in der Community auch das Unternehmen, in dem sie beschäftigt sind. In einem Fall, weil das Unternehmen aus der Koalition der Microsoft-Kläger ausgeschert ist, in andern Fällen, weil das Unternehmensprojekt, in dem „MSB" wichtig ist, von diesem nicht weiter verfolgt wird. Gelegentlich wird der/die EntwicklerIn auch von anderen Unternehmen abgeworben. Für die meisten Hauptentwickler steht die Arbeit an dem Projekt über längere Zeit im Mittelpunkt ihrer Berufslaufbahn und ihrer spezifischen Kompetenz, auch wenn sich die Einbindung und Finanzierung dieser Tätigkeit durch Unternehmen ändert. Diese geringe Fluktuation ist wahrscheinlich auch ein Grund dafür, dass die Software kontinuierlich entwickelt wurde und mittlerweile eine hohe Komplexität und einen hohen Reifegrad erreicht hat. Reine Hobbyentwickler finden hier nur schwer ein reizvolles Betätigungsfeld. Gelegentlich kommen Studenten hinzu, die für eine Arbeit im Rahmen ihres Studiums ein Thema aus der Entwicklung der Software wählen. Viele der neu hinzukommenden Entwickler tun dies allerdings unmittelbar aufgrund ihrer Aufgaben in einem der beteiligten Unternehmen. Sie gehören dann zu dem jeweiligen Unternehmensteam und dies prägt weitgehend die Beiträge, die sie in die Community einbringen.

Die Entwickler kommen mehrheitlich aus sechs bis acht Unternehmen, die ein strategisches Interesse an dieser Open Source Software haben. Es sind dies meist Unternehmen, deren wirtschaftliche Interessen unmittelbar mit der im Projekt entwickelten Software verknüpft sind, z.B. weil sie die Software für ihre eigenen Produkte weiterentwickeln oder weil sie Dienstleistungen rund um die OSS anbieten. Darüber hinaus gibt es gelegentlich auch Entwickler aus weiteren Unternehmen, die für eine gewisse Zeit involviert sind, z.B. Gerätehersteller, die eine bestimmte Funktionalität der OSS für ihre Gerätetreiber benötigen. Eine Vielzahl von Unternehmen sind reine Nutzer, die das entwickelte Softwareprodukt unverändert oder mit leichten Anpassungen in ihren betrieblichen Prozessen oder in ihren Produkten einsetzen, z.B. Hersteller von Speichergeräten und -systemen, spezialisierten Servern oder anderen Endgeräten. Auch diese verwenden zunehmend offene Softwareprodukte und IT-Dienstleistungen. Der Zugang zur Software ist durch die verwendete Lizenz (hier GPL V2/V3) grundsätzlich allen Anwendern offen. Um auf die für die Anpassung und Weiterentwicklung der „MSB" Software auf ihre besonderen Anforderungen erforderlichen Kompetenzen zuzugreifen, gehen sie meist einen von zwei We-

gen: manchmal engagieren sie sich (zeitlich begrenzt) mit eigenen Entwicklern, wesentlich häufiger finanzieren sie jedoch Entwickler aus der Community bzw. vergeben Aufträge an Unternehmen im Feld.

Uns interessieren im Rahmen unserer Fallstudie jene Unternehmen, die die Software im Rahmen ihrer eigenen Innovationsprozesse nicht nur einsetzen, sondern auch strategisch weiterentwickeln und dabei mit eigenen MitarbeiterInnen und Beiträgen konstant an der Entwicklung beteiligt sind. Mit einem Entwicklungsdienstleister (IT20) und einem Linux-Distributor (IT19) fokussieren wir in unserer Fallstudie auf zwei dieser Unternehmen[2].

7.4.2 Die Auswirkung gemeinschaftlicher Governance auf den Wissenstransfer

Im ersten Schritt soll der Wissenstransfer innerhalb der Community untersucht werden. Es soll untersucht werden, inwiefern die gemeinschaftliche Koordination des Entwicklungsprozesses innerhalb der OSS-Community den Wissenstransfer zwischen den beteiligten Akteuren beeinflusst. Gemäß der oben entwickelten Begriffsbestimmung orientieren wir uns bei der Untersuchung an den vier Dimensionen gemeinschaftlicher Governance in Form von OSS-Communities: technische Infrastruktur, kollektive Identität und gemeinsame Ziele, Institutionalisierung kollektiver Vorstellungen bzw. Normen kollektiven Handelns und interne Differenzierungs- und Organisationsprozesse.

7.4.2.1 Offenes Produkt und transparenter Produktionsprozess

Den wohl auffälligsten Aspekt des Wissensaustauschs innerhalb der Community bilden die durch die verwendete OpenSource-Lizenz sichergestellte Offenheit des Produkts und die durch die technische Infrastruktur bedingte Transparenz des Produktionsprozesses.

Die Open Source Software ist nicht nur als Quellcode frei verfügbar, sondern auch der Prozess ihrer Entwicklung in der Community ist für jeden Interessierten auf der Kollaborationsplattform im Web transparent nachvollziehbar. Jeder, der über entsprechende Kompetenzen verfügt, kann den aktuellen Stand und gegenwärtige Arbeiten an der Software (zurück-)verfolgen. Dies betrifft nicht nur die Möglichkeit, den aktuellen Bestand an geschriebenem Programmcode einzusehen, sondern aufgrund eines bereitgestellten Archivs der genutzten Mailingliste auch die Möglichkeit, die im Laufe der Entwicklung geführten Diskussionen zwischen den Entwicklern nachträglich nachzulesen und so z.B. die Gründe für getroffene Ent-

[2] Details zum Sample, den genutzten Untersuchungsmethoden und weiteren Quellen bietet das Methodenkapitel.

scheidungen nachvollziehen zu können. Durch die dauerhafte Archivierung der Beiträge und Bearbeitungsschritte lässt sich der Entwicklungsprozess über Jahre zurückverfolgen.

Auf der Basis der offenen Codebasis und des transparenten Entwicklungs- und Diskussionsprozesses kann sich eine offene Diskussionskultur entfalten, die wichtige Koordinationsfunktionen erfüllt, indem sie Austausch und Kooperation der involvierten Entwickler erleichtert. Das folgende Zitat ist von einer Entwicklerin, die hier ihren (Wieder-)Einstieg in die Community beschreibt:

> *„That's the beauty of Open Source. If you can program you take the source and you start talking. It's not always easy. But you just start talking. If you have a problem, you take the code, you create a fix for the problem and you propose it to the community, usually done by the mailing list. […] And of course you can ask technical questions. Chances are somebody will answer. […] The project is still completely public and community-based. Anyone can contribute."* (FS17-Entwickler-Community-1)

Dieser technisch gestützte, offene Entwicklungsprozess hat weitreichende Effekte für den Umgang mit dem in der Software verkörperten Wissen. So sind Entwickler in der Lage, beliebige Teile der Software zu verändern, zu ergänzen und weiterzuentwickeln, indem sie den bestehenden Code nehmen und weiterentwickeln – unabhängig davon, von wem er geschrieben wurde. Damit sind kompetente Anwender in der Lage, die Ursachen für unerwünschte oder fehlerhafte Funktionen der OSS selbständig zu erkennen und zu korrigieren. Durch die Rekonstruktion des Entstehungsprozesses besteht gerade für neu im Projekt beteiligte Entwickler die Möglichkeit, den Entstehungsprozess und die Entscheidungsgründe für bestimmte Design-Entscheidungen nachzuvollziehen, um die Logik des Codes besser erfassen zu können. Die Sichtbarkeit des Quellcodes und die Dokumentation der Erstellung schaffen damit sehr günstige Bedingungen für Lernprozesse der neuen Entwickler, die notwendig sind, um auf der Grundlage von bestehendem Code neue Funktionen oder Verbesserungen zu entwickeln.

Die (technisch vermittelte) Offenheit des Entwicklungsprozesses erleichtert den Wissenstransfer zudem dadurch, dass andere Entwickler die Möglichkeit haben, bereits in einem frühen Stadium der Entwicklung Interesse und Unterstützung anzubieten, wie das folgende Zitat verdeutlicht:

> *„Das ist das Wesen […] von Open Source, wenn es wirklich gelebt wird: ich muss mit dem, was ich tue, nicht ins Kämmerlein gehen und hinter mir zuschließen und wenn es fertig ist, gehe ich raus und verkaufe es, sondern ich kann auch währenddessen schon offen sein. […] Wenn ich meine Arbeiten […] voranbringe, dann mache ich das völlig öffentlich. Wenn ich irgendwas gemacht habe, schiebe ich die Änderungen auf mein privates […] hoch. Da steckt ja auch keine Geheimsache hinter. Und es passiert übrigens tatsächlich, dass auf der Mailingliste Leute auftauchen, die sagen, ist ja interessant, kann ich da irgendwas testen, kann ich irgendwas helfen. Das passiert."* (FS17-IT19)

Bezogen auf den *Wissenstransfer* können wir daher festhalten, dass mit dem offenen Code die expliziten Bestandteile des Wissens über die Grenzen der Community und der involvierten Unternehmen hinweg frei verfügbar sind. Dies betrifft ausdrücklich auch die Historie der Entwicklung und Diskussionen, die schriftlich auf den Mailinglisten geführt werden und von der verwendeten Entwicklungsplattform vorgehalten werden.

Es wäre jedoch sehr vereinfachend, die Community auf diesen technischen Aspekt und den Wissenstransfer auf den freien Zugriff auf gemeinsame Code-Teile zu reduzieren. Und auch wenn die Mailingliste (und das Archiv derselben) das zentrale Medium für den Austausch und die Koordination in der Community ist, gibt es darüber hinaus vielfältige Möglichkeiten der bilateralen Kommunikation zwischen den Entwicklern über E-Mail oder andere Medien, aber auch face-to-face Treffen der Entwickler z.B. auf der jährlichen Entwicklerkonferenz der Community. Wie sich in der Untersuchung deutlich zeigt, sind die sozialen Beziehungen in der Community zwar weitreichend technisch vermittelt, gehen aber keineswegs darin auf (vgl. auch Dolata & Schrape 2014). Gerade um zu verstehen, wie der Wissensaustausch innerhalb der Community funktioniert, ist es wichtig, die sozialen Praxen der Community zu untersuchen, die es den beteiligten Entwicklern ermöglichen, auch implizite Wissensbestandteile zu teilen.

7.4.2.2 Kollektive Identität, Institutionen und interne Organisationsprozesse der Community

Grundlegend für die gemeinsame Praxis der untersuchten Community ist das gemeinsame Ziel seiner Mitglieder, die MSB-Software als funktionsfähige Alternative zu Microsofts Netzwerkprotokoll zu entwickeln und zu verbreiten. Auf dieses Ziel sind alle Aktivitäten der Entwickler genauso wie die Organisation und Koordination des Produktionsprozesses der Software ausgerichtet. Die Beiträge der Entwickler in der Community sind wesentlicher Bestandteil ihres fachlichen und beruflichen Selbstverständnisses und ihrer Identität im Rahmen der Community wie in dem Unternehmen, in dem sie beschäftigt sind. Die Community verfügt gemeinschaftlich über das Open Source Produkt (durch die GPL Lizenz abgesichert) und etabliert Regeln und Normen für die Koordination der verteilten Arbeit, Positionen und Entscheidungsprozesse, d.h. sie bildet eine Organisationsform für die gemeinschaftliche Produktion.

Die Arbeitsverteilung in der Community ist selbstorganisiert und freiwillig. Daher gibt es auch keine Weisungsbefugnis einzelner Mitglieder anderen gegenüber. Eine wesentliche Triebfeder für Beiträge ist die Wahrnehmung von Bedarfen und Defiziten in der Community und darüber hinaus, sowie die Solidarität und Bereitschaft von einzelnen Entwicklern, die Probleme von anderen zu lösen. Solche Defizite können auf unterschiedliche Weise an die Entwickler herangetragen werden.

„Es war so eine Sache, der Schuh hat gedrückt und ich glaube, weil die Leute gesagt haben, hey du kannst doch WinBind, wir haben hier einen komischen WinBind-Bug und ich hab mir den angeguckt und der sah auch nicht so besonders toll aus. Also ich habe mit [Name entfernt – PF] zusammen gesessen und er hat nebenher noch ein bisschen was anderes gemacht und herübergeguckt und ein paar Tipps gegeben und dann haben wir halt innerhalb von einer Woche diesen DNS-Server da hingestellt. Weil es aussah, als ob man den braucht […] Das war jetzt nicht, dass irgendjemand vorher gesagt hat, das brauchen wir dringend, sondern man hat halt gesehen, ok hier drückt der Schuh gerade und ich hab gerade Zeit, dann mach ich das einfach mal." (FS17-Entwickler-Community-2)

Neben der fehlenden Weisungsbefugnis existieren auch keine direkten Sanktionsmöglichkeiten, durch die Tätigkeiten von Mitgliedern erzwungen werden könnten. In begrenztem Umfang haben sich informelle Regeln herausgebildet, die soziale Erwartungen konstituieren, wie z.B. bei der Behandlung von Fehlern oder der Bearbeitung von Fragen bezogen auf die Teile der Software, die man selbst geschrieben hat. Die gemeinsamen Ziele bezogen auf die Software, sowie gemeinschaftliche Werte wie Solidarität und Aufmerksamkeit für Anforderungen anderer sind in der Community entscheidende Faktoren, die die Bereitschaft zur aktiven Mitarbeit befördern. Dabei spielen natürlich auch die Wahrnehmung der eigenen Kompetenz und die soziale Anerkennung und Zuweisung von Kompetenzen eine Rolle. In diesem Zusammenhang können soziale Beziehungen, Sympathie und die Erfahrung angenehmer, fruchtbarer Zusammenarbeit eine starke Bedeutung gewinnen.

„Es passiert ja nichts wenn ich es nicht tue. Ich hab ja keinen Chef, der mir auf die Finger haut, wenn ich irgendwas nicht tue. Das trifft halt bei der ganzen Open Source-Entwicklung viel, viel stärker zu, dass man die Aufmerksamkeit der Leute sich ganz anders erarbeiten muss. Man kann nicht mit Geld und irgendwelchen Sanktionsmöglichkeiten etwas erzwingen. Sondern man muss eben die Aufmerksamkeit der Leute erreichen. Und es zwingt mich ja niemand, irgendwelche Patches, die mich nicht interessieren, anzugucken. Und das ist schon deutlich anders. Die Freiheit ist da sehr groß. Man muss einen guten Sparrings-Partner haben und das Verhältnis muss gut sein. Wenn mich irgendjemand permanent nervt, dann hab ich auch keinen Bock, seine Patches zu reviewen. Und dann tue ich es auch nicht. Das hat ja keine Konsequenzen. Klar, […] wenn jetzt irgendwie jemand, der mich furchtbar nervt, mit einem schweren Sicherheitsproblem in „MSB" kommt, dann betrifft mich das auch, dann werde ich natürlich auch drauf springen. Selbst wenn der Typ mich nur nervt […]." (FS17-IT20-2)

Da es keine hierarchische Verteilung von Aufgaben gibt, haben Reziprozitätsnormen in der Entwickler-Community eine große Bedeutung für die Koordination der Arbeit und die Zusammenarbeit in der Community, wie das folgende Zitat veranschaulicht:

„Da kommen wir wieder zu der Sozialstruktur des „MSB"-Teams. Was ich mache hängt einerseits an der reinen Technik. Andererseits aber auch daran, wie mein Draht zu den Leuten ist. Dann ergibt es sich natürlich, dass man da sagt, hier ich hab da mal eine Frage, kannst du mir da mal helfen. Dann passiert das natürlich schon mal eher als mit jemanden wo ich persönlich gar nicht kann. […]

Frage: Und nach dem Muster, also ich mach jetzt was dir wichtig ist, und du machst im Gegenzug dann etwas für mich?

Das wiederum passiert. Also das passiert insbesondere in diesem Review-Bereich. Also das ist schon so ein bisschen, eine Hand wäscht die andere." (FS17-IT20-2)

Dies bedeutet jedoch nicht, dass alle gleichermaßen über den Fortgang der Entwicklung bestimmen würden. So gibt es durchaus eine soziale Zuweisung und Wahrnehmung von Verantwortlichkeiten, die auf der zentralen Institution der Community basiert: Ähnlich wie in wissenschaftlichen Produktionsgemeinschaften (Gläser 2006) gibt es auch in der „MSB"-Community eine persönliche *Autorenschaft* der Entwickler, die sich auf den Code bezieht, dass sie (mit-)entwickelt haben: Wer ein Stück Code geschrieben hat, das nachhaltig in das Programm aufgenommen wurde, fühlt sich hierfür verantwortlich, wie der folgende Entwickler ausführt:

„Dabei ist natürlich ein wichtiger Punkt: also der DNS-Server ist meiner! Ich fühl mich natürlich auch verantwortlich wenn irgendjemand jetzt kommt und sagt: oh, hier ist ein Bug! Dann sag ich: oh, hoppla, den reparier ich mal lieber! Weil, wie gesagt, ich hab's verbrochen, ich fühl da auch eine gewisse Verantwortlichkeit dem gegenüber." (FS17-Entwickler-Community-2)

„Autorenschaft" kann also durchaus als wichtige Institution innerhalb der Community angesehen werden, sie ist ein zentrales Prinzip der Koordination, das zwar informelle aber nichtsdestotrotz wirksame Zuweisungen von Verantwortlichkeiten beinhaltet, wie das folgende Zitat beschreibt:

„Wer schreibt, der bleibt. Also wenn jemand an der Software was ändert, wenn das was Neues ist, dann hat der gleichzeitig eine Aufgabe. Weil er oder sie ist dann nachher auch dafür zuständig. Der steht in der Copyright-Liste vorne als Autor. Es gibt im „MSB"-Team nur persönliche Copyrights, es gibt keine Corporate Copyrights, das heißt, es gibt sozusagen immer nur den Mensch, der das Projekt gestartet hat und die anderen schreiben sich oben drauf. Aber es gibt für jedes Stück Programm immer einen, der das mal angefangen hat. Und das bedeutet, wenn dann einer sagt, ich mache was Neues, dann hat der auch die Zuständigkeit. Dann wird jemand, der da was reinschreibt, sich mit ihm abstimmen. Das heißt aber nicht, dass jemand das nicht darf. Sondern es ist dann nur Konvention, dass man sich dann abstimmt und dass man nicht in den Code von anderen Leuten reinschreibt. Es gibt immer riesen Geschrei im Team und Flamewars im Team, wenn jemand bei jemand anderem was reinschreibt und einfach Sachen ändert von anderen Leuten, ohne sie zu fragen. Das sind Abstimmungsprozesse, richtig schwer. Von lauter unabhängigen Leuten. […] Also das ist so ein typisches Community-Thema. Aber

normalerweise ist der Prozess so, da gibt es Zuständigkeiten und man stimmt sich in einer freundlichen Umgebung ab." (FS17-IT20-1)

Der Nachvollzug der Autorenschaft wird durch die technische Kollaborationsplattform unterstützt, in der nicht nur der Code dokumentiert ist, sondern für jede Code-Zeile auch der Entwickler festgehalten wird, der sie geschrieben hat:

„Also unser Versionskontrollsystem GIT kann an jeder Codezeile sagen: wer hat das zuletzt angefasst. Das ist einfach ein Basismechanismus der Versionskontrolle. Ich will immer nachgucken können, wer war es. Wen kann ich im Zweifel fragen, warum er das so gemacht hat. Wem kann ich auf die Finger hauen, wenn er Mist gebaut hat im Nachhinein. Das ist auch ganz entscheidend für irgendwelche Urheberrechtsgeschichten. Wir müssen nachvollziehen können, wer als Autor zeichnet verantwortlich für eine gewisse Zeile Code, wenn es mal zu irgendwelchen Urheberrechtskonflikten kommen sollte. Und die hat es gegeben in der Vergangenheit, diese Fälle." (FS17-IT20-2)

Autorenschaft konstituiert somit wechselseitige soziale Verpflichtungen: Entwickler sollen die Arbeit vorheriger Autoren respektieren und daran anschließen, indem sie dem Autor ein Mitspracherecht bei Änderungen einräumen. Der Erstautor hat letztlich sogar ein Vetorecht, wenn er dieses fachlich begründen kann. Auf der anderen Seite wird von Autoren erwartet, dass sie auf Fragen und Änderungsvorschläge anderer Entwickler antworten und diese unterstützen, indem sie ihre Erfahrungen bei der Bearbeitung dieses Programmabschnitts teilen. Sie konstituiert die bereits beschriebene Verantwortlichkeit für das Stück Code, sowohl als Verpflichtung bei der Beseitigung von Fehlern als auch als Einfluss auf diejenigen, die hier Änderungen durchführen wollen. Autorenschaft gewährleistet darüber hinaus den persönlichen Kontakt und Austausch der Entwickler mit dem Erstbearbeiter bzw. den Erstbearbeitern des konkreten Programmteils, an dem er oder sie gerade arbeitet. Der Austausch geht oft so weit, dass der Erstautor konkrete Hinweise zur Programmierung gibt, wodurch die neue Person wichtige Einblicke in nicht explizierte Vorgehensweisen und Techniken gewinnen kann:

„Und das ist aber eben der schöne Nebeneffekt, dass wir dann immer sehr genau wissen, wen kann ich fragen, warum irgendwas so funktioniert, wie es denn funktioniert. […] Nennt sich GIT-Blame, das Kommando. GIT-Blame auf eine Datei – an jeder Zeile steht drin, das ist jetzt von dem und dem zuletzt geändert worden. Und dann kannst du auch sehr genau die Historie nachverfolgen, wie sah es vor der Änderung aus, wer hat die Finger da drin gehabt. Also das ist schon sehr ausgefeilt. Dieses sogenannte verteilte Versions-Kontrollsystem ist sicherlich eine reine Technik. Eine rein technische Lösung für ein soziales Problem sozusagen. Nämlich, ich will mich viel freier mit Leuten austauschen können über irgendwelche Codestücke, ich möchte das nicht alles über irgendeinen zentralen Server machen müssen." (FS17-IT20-2)

Aus der Verantwortlichkeit für bestimmte Teile des Programms entwickeln sich dadurch innerhalb der Community *informelle Teamstrukturen* von Entwicklern, die zusammen an einem bestimmten Programmteil arbeiten. Sie kooperieren auf der Basis von Reziprozität und oft entwickelt sich aus dieser dauerhaften gemeinsamen Arbeit auch eine persönliche Vertrautheit und dauerhafte Beziehung. Auf diese Weise wird persönliches Erfahrungswissen der Entwickler anerkannt und für die Weiterarbeit systematisch einbezogen. Durch die relativ dauerhaften Strukturen und Aufgabenfelder entsteht zudem ein eigener Arbeitsbereich mit einer gemeinsamen Praxis, die durch die vertraute Beziehung und engen Kontakt auch den Austausch von implizitem Wissen nachhaltig fördert.

Die Hauptentwickler geben ihre Erfahrungen und ihr Wissen in vielfacher Weise weiter, teilen Erfahrungswissen und unterstützen sich gegenseitig. Diese Verhaltensorientierungen entwickeln sich in der gemeinsamen Praxis zwischen den Entwicklern, ohne dass sie formal fixiert sind. Dabei lassen sich unterschiedliche Grade des Austauschs feststellen. Der informelle Austausch von Erfahrungswissen und die gegenseitige Unterstützung bei Problemlösungen ist am intensivsten zwischen Entwicklern, die auch unmittelbar bei der Programmentwicklung zusammen arbeiten, zwischen denen in fachlicher und sozialer Hinsicht eine Vertrauensbeziehung gewachsen ist:

> *„Kooperation findet auch schon viel, viel früher statt [vor dem review prozess – PF]. Also einfach, ich sehe ein Problem und ich hab von irgendwas begrenzt Ahnung, dann frag ich einfach meine Kollegen. Oder, ja das findet viel früher statt. Irgendwann hat man halt raus, wer sich wofür zuständig fühlt, wer wovon Ahnung hat, und dann kann man halt eben entsprechend fragen. Und dann wechseln Ideen quasi hin und her."* (FS17-IT20-2)

Gleichzeitig gibt es auch einige stärker formalisierte Formen des Erfahrungs- und Wissensaustauschs, wie z.B. den Reviewprozess von Patches (neuem Code), in dem entschieden wird, ob ein neues Stück Code in die Community-Software aufgenommen wird.

> *„Dieser Reviewprozess ist sicherlich ein konkreter Punkt, an dem es dann auch ein Stück weit formalisiert ist. […] Patch-Review heißt ja, ich habe etwas von dem ich glaube, dass es fertig ist. Ich stelle es zur Verfügung und dann soll jemand noch mal darüber gucken."* (FS17-IT20-2)

Dabei wird im Vier-Augen-Prinzip auch teamübergreifend über neue Codebestandteile entschieden, bevor neuer Code eingepflegt wird:

> *„Was wir gemacht haben ist, wir haben einen sogenannten Reviewprozess eingeführt. Das ist im Moment auch noch nicht wirklich formal, aber jeder hält sich dran. Bevor irgendeine Änderung, ein Patch reinkommt in „MSB", müssen zwei Leute vom „MSB"-Team zustimmen. Das haben wir inzwischen mit so kleinen Tags, mit einem Commitment-Switch versehen, das heißt natürlich, wenn ich jetzt als „MSB"-Team-Mitglied etwas entwickle, muss ich einen anderen finden. […] Aber das hat sich eigentlich als ganz gut erwiesen.*

Und da ist es natürlicherweise so, dass wenn jemand was am Fileserver drehen will, dass ich dann bevorzugt mir diese Patches angucke. Das ist nirgendwo formalisiert. Aber es gibt schon da Leute, die sich eben bevorzugt für einzelne Felder interessieren und dann eben auch draufspringen." (FS17-IT20-2)

Im Reviewprozess scheint der erfahrungsbasierte Charakter des nötigen Wissens deutlich auf. Obwohl der Prozess an sich recht formalen Kriterien genügt, ist die Prüfung der Qualität des Codes wenig formalisiert, weil es kaum formalisierte und explizit formulierte Qualitätskriterien gibt. Dabei geht z.B. um die Frage, ob beim Einfügen eines neuen Codeteils auch im Kontext des neuen Codes Anpassungen vorgenommen werden müssen, ob der neue Code sinnvoll platziert ist oder ob er den impliziten „Designregeln" entspricht. Im folgenden Zitat wird deutlich, das Designregeln und Konventionen im Entwicklungsprozess entwickelt und etabliert werden und sich nur höchst unvollständig aus dem explizierten Code selbst erschließen lassen:

„Klar, ich versuche als Entwickler immer, selber Probleme zu lösen. Ich denke, ich würde es irgendwie gelöst kriegen, nur kann es halt sein, dass es nicht mit dem Design von „MSB" übereinstimmt. Und da ich quasi nicht in die Design-Entscheidungen von dem Active Directory-Code eingebunden bin, frage ich halt andere, passt das in euer Design rein? Passt das in eure grundsätzlichen Ideen, wie das zu funktionieren hat, sauber rein? Soll es so sein? Und das sind dann halt relativ frühe Entscheidungen, wo ich dann andere frage. Von denen ich weiß, dass sie sich damit besser auskennen." (FS17-IT20-2)

Auch die folgende Aussage macht auf den prozessoralen und diskursiven Prozess aufmerksam, der zu gemeinsam geteilten Auffassungen bzgl. Konventionen und Qualitätsstandards führt:

„Wir haben aber sehr viele, sehr hübsche elegante Subsysteme im „MSB" im Laufe der Zeit entwickelt und aufgesammelt, die es zu einem sehr, sehr angenehmen Arbeiten machen. Wir haben z.B. keine Objektorientierung in C. Was wir aber dadurch ersetzen, dass wir Abstraktionsschichten einziehen. […] Das ist so ein typisches Muster, was man immer wieder mal antrifft. Also dass man eine Funktion in eine Funktion reinreicht, um dem Allgemein-Gerüst einen speziellen Anstrich zu geben. Das sind so Sachen, wenn man es verstanden hat, denkt man, so soll es sein, das ist elegant, das ist toll. Und da sind sich im Prinzip alle einig. […] An sich geht es immer eher so in die Richtung, dass man eine Idee mit den Leuten bespricht und Feedback kriegt. Die anderen sagen, das hast du gut gemacht oder das kann noch besser sein. Also eigentlich ein sehr konstruktives Zusammenarbeiten. Das ist der Grundtenor." (FS17-IT20-3)

Um entscheiden zu können, welche Konventionen und Qualitätsstandards der Community bei der Entwicklung eines Patches[3] berücksichtigt werden sollten, ist häufig implizit bleibendes Erfahrungswissen gefordert. Zwar gibt es in der Community einige schriftlich festgehaltene Guidelines und Regeln, häufig sind solche Konventionen aber nicht formell festgeschrieben, sondern es handelt sich um geteilte Ansichten über „guten Code" und „gutes Design". Diese sind z.T. in professionellen Standards, wesentlich häufiger aber in spezifischen Erfahrungen aus dem „MSB"-Erfahrungsschatz begründet:

> *„Ein Beispiel: Wenn wir intern in „MSB" Hauptspeicher managen müssen. Die Programmiersprache C hat dafür ein Mittel, Hauptspeicher bereit zu stellen. Diese Routine nennt sich Malloc. Malloc ist nicht mehr schön, das ist erstens langsam und das ist zweitens fehlerträchtig. Wir haben Talloc entwickelt, was potentiell sehr viel schneller ist und nicht fehlerträchtig. Deswegen bitteschön, benutzt doch Talloc. Es gibt auch heute noch Code in „MSB", der Malloc benutzt. Neuer Code würde das nicht tun. Das ist eine Entwicklung der letzten zehn Jahre ungefähr. Es entwickeln sich quasi im Code irgendwelche Konventionen, das tut man einfach nicht mehr. Aber das ist nicht immer offensichtlich, solche Sachen."* (FS17-IT20-2)

Diese geteilten Ansichten, Konventionen und Regeln werden diskursiv eingefordert und spätestens im Zuge des Reviews durchgesetzt. Denn die Reviewer können durch ihren Einspruch verhindern, dass ein Patch in der vorliegenden Form in die offizielle Codebasis aufgenommen wird:

> *„Wir haben in „MSB" ein readme.coding. Da stehen unsere Coding-Guidelines drin. Und alles, was da nicht drinsteht, da gibt es dann Diskussionen drüber. Vieles steht nirgendwo wirklich konkret niedergeschrieben und das wird dann in einem Patch-Reviewprozess diskutiert. Und dann gibt es Diskussionen darüber und entweder derjenige fügt sich dann oder sieht es ein oder der ist dann beharrlich und stur genug, es doch durchzuziehen. Gibt halt niemanden, der wirklich sagt, so und nicht so."* (FS17-IT20-2)

Die Diskussion über solche Konventionen für die „MSB" Entwicklung haben für Entscheidungsprozesse in der Community deswegen eine besondere Relevanz, weil Entscheidungen in der Community im Prinzip auf Konsensfindung und Abstimmung unter den Mitgliedern beruhen. Wichtige Entscheidungsprozesse in der Community betreffen dabei vor allem die Produktentwicklung, d.h. jede Veränderung der veröffentlichten Version der Software setzt einvernehmliche Entscheidungen der Mitglieder der Community voraus. Es geht dabei vor allem um die Integration von vorgeschlagenen Patches, in die öffentlich verteilte Softwareversion. Daneben müssen auch grundsätzliche Entscheidungen getroffen werden, wie z .B. die Frage, welche Version der GPL für die Community Software insgesamt angewendet wird.

[3] Ein Patch ist eine Erweiterung des Programmcodes, häufig, um ein spezifisches Problem zu lösen oder eine neue Funktionalität bereitzustellen.

Für diese Entscheidungsprozesse haben sich Regeln und Konventionen herausgebildet. Letztlich werden Entscheidungen mehrheitlich über Abstimmung auf der Mailliste getroffen. Nicht immer kommt es zu formellen Entscheidungen, in die die Mehrheit der Mitglieder involviert ist. In vielen Fällen wird eine Entscheidung vorgeschlagen und wenn es keinen Widerspruch gibt, gilt sie als angenommen. Abstimmungen sind nur nötig, wenn es Widerspruch gibt. In solchen Abstimmungen haben die Autoren und die Gründungsmitglieder ein Vetorecht, d.h. deren Gegenargumente müssen entkräftet bzw. widerlegt werden.

Die persönlichen Beiträge der Entwickler sind dabei die zentrale Einheit bei der Zuweisung sozialer Anerkennung und entsprechendem Einfluss in der Community:

> *„Also das ist quasi so eine Art natürliche Autorität, die sich da entwickelt. Weil man einfach seit 20 Jahren das Zeug macht, hat man auch einfach relativ schnell aus dem Rückenmark irgendwelche Argumente, warum das eine gute Idee ist und das keine gute Idee ist. Weil man einfach in viele Fehler selber schon reingelaufen ist, deswegen kann man halt relativ schnell eben irgendwelche Vorschläge – guck mal da, das haben wir da aber schon gemacht, hat nicht funktioniert – zum Beispiel wenn ein Patch halt schlecht ist, wegdiskutieren. Und insofern, das ist schon recht effizient, man guckt sich das an, man sieht, das interessiert mich, und man springt da rein. Und das funktioniert eigentlich ganz gut. Wenn die Leute sich aktiv an diesen Diskussionen beteiligen, ergibt sich sowas wie eine natürliche Autorität. Das ergibt sich einfach. Ohne dass man da jetzt irgendwie ein formales Organigramm dafür bräuchte."* (FS17-IT 20-2)

Die soziale Anerkennung für akkumulierte, geleistete Beiträge manifestiert sich auch darin, dass Ranglisten geführt werden, die zeigen, wer in welchem Zeitraum viele Beiträge eingebracht hat (z.B. auf der Plattform Ohloh.org). Soziale Anerkennung in der Community ist ein zentraler Faktor für die Motivation der Entwickler, aber auch für ihre berufliche Karriere. Wichtige Entwickler aus der Community haben häufig keine Probleme, eine angemessene Beschäftigung bei einem Unternehmen im Feld zu finden und dort eine gute Position zu erlangen. Denn eine wichtige Position in der Community gilt als Nachweis für hohe fachliche Kompetenz, denn das fachliche Niveau in dieser Community wird auch außerhalb der Community anerkannt.

Wie sich im letzten dokumentierten Zitat deutlich zeigt, gibt es in der Community starke meritokratische Einflüsse: Wer viel einbringt, kann darauf setzten, dass er/sie in der Community mehr Einfluss bekommt. Formale Abstimmungen finden zwar auch statt, aber sie sind überlagert von meritokratischen Positionen:

> *„Dann gibt es aber den Effekt in diesen Communities, dass natürlich Leute gleicher sind, die schon ein bisschen länger dabei waren. D.h. wenn [Name entfernt – PF] und [Name entfernt – PF] sagen, wir wollen das so, dann passiert das meist auch so. Und das Problem ist, dann gab es jede Menge Diskussion und dann gab es zum Teil schon Abstimmungen, mit einem Ergebnis, das dann aber den Ältesten nicht passte und dann wurde das wieder neu aufgerollt und geändert."* (FS17-IT20-1)

Doch trotz dieser meritokratischen Struktur kann keiner der Hauptentwickler eine Entscheidung gegen die anderen erzwingen, sondern im Konfliktfall muss in einem mehr oder weniger heftigen Diskussionsprozess ein Kompromiss gefunden werden.

> *„Ich habe nicht den Eindruck, dass es dieses Durchgreifen gibt. Also es gibt Leute, die wirklich viel machen, was auch wirklich gut ist, die eine Meinung haben, die mehr Gewicht hat. Der Meinung von neueren Teammitgliedern, jemand der sich erst mal im Rahmen vom Team beweisen muss, wird vielleicht nicht ganz so viel Gewicht gegeben. Also der muss mehr argumentieren, als jemand von den alten Hasen. […] Aber, ich hab nicht den Eindruck, dass es irgendjemanden gibt, der wirklich in Anspruch nehmen kann, wirklich alleine aufgrund seiner Position im Projekt eine Entscheidung zu treffen, die nicht einem bestimmten technischen Bereich zuzuordnen ist. Also wir haben viele Bereiche vom Code, wo irgendjemand, weil er da ganz viel dran arbeitet, die Person ist, die Autorität über diesen Code hat."* (FS17-Entwickler-Community-2)

Dies kann ein langwieriger und schmerzhafter Prozess sein. Entscheidungsprozesse in der Community können so erschwert und verlangsamt werden, wenn es zu bestimmten Punkten abweichende Meinungen gibt:

> *„Diskussionen werden manchmal doch sehr emotional. Also ich hab den Eindruck, eine Menge von den schwierigen Entscheidungen sind deswegen schwierig, nicht weil irgendwie die technische Umsetzung davon schwierig wäre, sondern weil es halt doch von Leuten gemacht wird, und das heißt man muss eigentlich immer Egos navigieren und seltener wirklich die technische Herausforderung alleine."* (FS17-Entwickler-Community-2)

Wie kompliziert solche Aushandlungsprozesse zwischen den Entwicklern sein können, zeigte sich an einem besonders gravierenden Beispiel eines über lange Zeit schwelenden Konflikts innerhalb der Community über das technische Design des Produkts, der sich schließlich in zwei Versionen des Produkts niederschlug. Die Community war annähernd 50:50 gespalten, was einen lang anhaltenden Patt zur Folge hatte, der die Arbeit der Community lähmte und fast zur Spaltung der Community geführt hätte. Erst die engagierte Vermittlung durch einige Community-Mitglieder führte damals schließlich zu einem Kompromiss und einem erneut gemeinsamen Vorgehen. Wir werden auf die in diesen Abstimmungsprozessen liegenden Probleme für die beteiligten Unternehmen unten noc h näher eingehen.

Wie in diesem Abschnitt argumentiert, beinhaltet die Community eine für den Austausch von Wissen förderliche soziale Praxis, die durch die gemeinsame Identität und geteilte Zielsetzungen, interne Institutionalisierungsprozesse geteilter Normen und Vorstellungen und interne Organisations- und Differenzierungsprozesse geprägt ist. Zusammen mit der technischen Infrastruktur der Entwicklungsplattform ermöglicht diese Praxis, dass verschiedene Akteure mit ganz unterschiedlichen Erfahrungs- und Wissensbeständen gemeinsam ein Produkt (weiter-)entwickeln. Allerdings ist der Fokus der vorliegenden Untersuchung nicht der Wissensaustausch per se, sondern die Frage, wie es den beteiligten Unternehmen gelingt, das in der Community generierte Wissen im Rahmen ihrer eigenen Innovationsprozesse zu

nutzen. Die Fragestellung hat damit auch eine strategische Dimension, die sich explizit darauf bezieht, inwiefern Unternehmen die Generierung und Nutzung des Wissens in der Community strategisch beeinflussen können. Und gerade diese Dimension der strategischen Steuerbarkeit ist es, die für die untersuchten Unternehmen in Bezug auf Community-basierte Innovationsprozesse zur größten Herausforderung wird und für die sie geeignete Lösungen finden mussten.

7.4.3 Steuerungsprobleme der beteiligten Unternehmen

An der Entwicklung von „MSB“ waren im Verlauf der vergangenen 20 Jahre zahlreiche Unternehmen beteiligt. Im Kern handelt es sich um sechs Unternehmen, die seit mehr als zehn Jahren jeweils durch mehrere Entwickler beteiligt sind. Diese Unternehmen haben ein strategisches Interesse an der Entwicklung von „MSB“, ihre Geschäftsmodelle beziehen sich auf diese OSS und sind bezogen auf das betreffende Geschäftsfeld (alle haben auch andere Geschäftsfelder) von der Entwicklung der Software abhängig.

Unternehmen 1 ist ein Entwicklungsdienstleister, der seinen Kunden Entwicklungs- und Supportdienstleistungen zu der Open Source Software „MSB“ anbietet. Das Geschäftsmodell besteht im Kern darin, das exklusive Wissen und die (durch die Community) ausgewiesenen Kompetenzen der hier beschäftigten Kernentwickler in der „MSB“ Community für Entwicklungs- und Supportdienstleistungen im Kontext der OSS „MSB“ zu vermarkten. Seine Kunden sind Unternehmen, die „MSB“ verwenden, entweder im operativen Einsatz im eigenen Unternehmen oder in ihren Produkten (in Hardware, Software oder IT-Services). Die Aufträge umfassen ein weites Spektrum: von der Problemlösung im operativen Einsatz, kleineren Entwicklungsaufträgen zur Anpassung an besondere Anforderungen bis hin zu großen und z.T. sehr offenen Entwicklungsaufträgen, die Entwickler über viele Monate oder sogar Jahre finanzieren. Das Unternehmen ist mit mehreren Hauptentwicklern beteiligt, die für eine hohe Zahl an Beiträgen zur offiziellen „MSB“-Version der Community verantwortlich sind.

Nahezu alle, auf jeden Fall alle wichtigen Aufträge von Kunden, werden – nachdem sie an die Kunden ausgeliefert wurden – so aufbereitet, dass sie als Beitrag für die Weiterentwicklung von „MSB“ eingereicht werden. Da die Kundenaufträge auch umfangreichere Entwicklungsaufträge umfassen, sind wesentliche Teile der Weiterentwicklung von „MSB“ „kundengetrieben“. D.h. Anwenderunternehmen beauftragen IT20, bestimmte, z.T. auch innovative Features für „MSB“ zu schreiben. Diese Features können sie über ihre Produkte selbst vermarkten (schneller als andere, aber nicht exklusiv) und sie können gleichzeitig davon ausgehen, dass die Entwickler diese Features über ihren Einfluss in der Community-Struktur auch in die offizielle „MSB“-Version einbringen. Dies gewährleistet, dass die Produkte auch mittelfristig mit der offiziellen Version von „MSB“ kompatibel sind. Gleichzeitig werden die Features, wenn sie Teil der offiziellen Version sind, auch von der Community gepflegt und weiterentwickelt.

Unternehmen 2 ist ein Linux Distributor, also ein Unternehmen im Linux Ecosystem, das die Open Source Software zu einem eigenständigen Linux-System mit zusätzlichen Serviceleistungen bündelt und vertreibt. Es gibt eine Basis-Version des Linux-Systems, die kostenfrei verbreitet wird, und kommerzielle Versionen, die kostenpflichtig sind und zusätzliche Leistungen, wie Anpassungen und Support beinhalten. „MSB" ist gegenwärtig integraler Bestandteil der Linux-Distribution. Das von uns untersuchte Unternehmen (IT19) ist eines der größten im Linux-Ecosystem. Es entwickelt stetig neue Funktionalitäten für „MSB" im Rahmen der eigenen, innovativen Entwicklungsstrategie, die über die Community in die offizielle „MSB"-Version eingebracht werden, die Software erweitern, verbessern helfen und damit gleichzeitig den Einfluss des Unternehmens und seiner Entwickler in der Community stärken (siehe unten). Für die Linux Distributoren ist es wichtig, dass das eigene Produkt kompatibel bleibt – sowohl innerhalb des Linux Ecosystems wie auch zur Windowswelt.

Der im Rahmen dieser Studie untersuchte Linux Distributor stellt mittlerweile das größte unternehmenseigene Entwicklerteam in der Community. Bezogen auf die Entwicklungstätigkeit sind die Beiträge dieser Entwickler zu „MSB" davon abhängig, wie groß die Überschneidungen des unternehmensinternen „Produktes" bzw. Arbeitsbereichs mit „MSB" sind. Hier gilt, dass die Arbeit der Entwickler am „Unternehmensprodukt" Vorrang hat. Aber es werden systematisch Überschneidungen zu „MSB" gesucht, oder anders formuliert, die eigene Softwareentwicklung soll so weit wie möglich in „MSB" eingebracht werden (strategische Interoperabilität, gemeinsame Pflege, Teilen der Entwicklungsarbeit). Dies hat zur Folge, dass auch umfangreiche, innovative Features mit vergleichsweise hohem Arbeitsaufwand vom Distributor für „MSB" entwickelt werden. Für die Entwicklungsperspektive von „MSB" ist dieses Unternehmen daher höchst relevant.

Trotz unterschiedlicher Geschäftsmodelle haben die beiden Unternehmen drei zentrale Gemeinsamkeiten. *Erstens*, das strategische Interesse daran, dass diese Software Open Source ist und interoperabel in Linux und Windows Umgebungen einsetzbar ist. Beides ist zumindest bezogen auf die von „MSB" abgedeckten Funktionalitäten konstitutiv für ihr jeweiliges Geschäftsmodell. *Zweitens*, sind sie nicht an allen Teilen der Software gleichermaßen beteiligt, sondern konzentrieren sich auf die Entwicklung der Funktionalitäten, die für ihr Geschäftsmodell relevant sind (dies muss nicht in gleicher Weise für die bei ihnen beschäftigen Entwickler gelten, die hier Spielräume haben und auch ihre eigene Biografie mit den daraus resultierenden Interessen und Verpflichtungen). Und damit zusammenhängend, sind sie *drittens* weder daran interessiert noch in der Lage, das gesamte „MSB" zu entwickeln.

Die beteiligten Unternehmen befinden sich also in einem schwierigen Spannungsverhältnis gegenüber der Community: Einerseits ist die Entwicklung der Software vom selbstorganisierten und -gesteuerten Entwicklungsprozess der Community abhängig, andererseits verfolgen die Unternehmen ökonomische Ziele, die in der Community zunächst systematisch keine Rolle spielen. Dieser Konflikt führte in

unserer Fallstudie zu zwei Problemkomplexen, die von den beteiligten Unternehmen bearbeitet werden mussten. Im Folgenden erläutern wir zunächst die zwei Hauptprobleme (Unmöglichkeit der zeitlichen Planung des Gesamtentwicklungsprozesses und Probleme der strategischen Ausrichtung des Entwicklungsprozesses), auf die wir in unserer Forschung gestoßen sind, bevor die jeweiligen Lösungen präsentiert werden, mit denen die Untersuchungsunternehmen auf diese Probleme zu reagieren versuchen.

7.4.3.1 Unmöglichkeit der zeitlichen Planung des Entwicklungsprozesses in der Community

Charakteristisch für Community-basierte Entwicklungsprozesse ist, wie oben bereits in Bezug auf zentrale Entscheidungsprozesse diskutiert wurde, dass es keine zwingende Möglichkeit gibt, eine Planung oder Roadmap festzulegen und umzusetzen. Dies hat zwei Gründe, zum einen verfügt die Community nicht über kalkulierbare Personalressourcen, sondern ist immer von der individuellen Bereitschaft von Entwicklern und deren oft begrenzten Zeitkapazitäten abhängig. Die meisten Entwickler der Community sind bei einem Unternehmen beschäftigt, ihre für „MSB" verfügbare Zeit ist daher auch vom häufig wechselnden und nicht einschätzbaren Arbeitseinsatz bei den jeweiligen Unternehmen abhängig. Zum anderen sind die Entwickler in der Community grundsätzlich frei in ihrer Entscheidung, wann und woran sie mit welcher Intensität arbeiten. Ein Entwickler skizziert dieses Problem folgendermaßen:

> *„Und es gibt keine ganz klare Roadmap. Roadmap heißt ja auch immer, mit einer zeitlichen Vorstellung, mit Milestones und so. […] Ich sag mal so, wir haben eine Liste von Features, die wir gerne entwickeln möchten. Zum Teil ergibt sich sinnvollerweise eine zeitliche Abfolge, aber ohne zu sagen, dann und dann muss es fertig sein. Das können wir nicht. Welche Reihenfolge genommen wird, das ist im Endeffekt den Leuten überlassen. Welche Wichtigkeit da ist, wird dann häufig von dem Arbeitgeber derjenigen bestimmt, die daran arbeiten. Es gibt kein Gremium das sagt, die und die Sachen werden gemacht im „MSB"-Team. Es ist mehr so, jemand sagt, ich werde daran arbeiten: ah ja, interessant, oh, aber bedenke, wenn du das machst dann will ich gerne damit zu tun haben. Und solche Reaktionen gibt es üblicherweise. Also es ist sehr offen, keine Hierarchie in dem Sinne. Kein Gremium, was das bestimmt. Wo es erst abgesegnet werden muss vom „MSB"-Team, damit jemand wirklich das Zeug reinbringen kann, das gibt es in der Form nicht. Wenn da genug Druck ist von irgendeiner Seite, dann wird das auch gemacht."* (FS17-IT20-3)

Es sind die zentralen Eigenschaften der Community, die oben als förderlich für den Wissensaustausch hervorgehoben wurden, die zu diesem Problem beitragen. Alle Planungen der Entwicklungsarbeiten sowie die Festlegungen der nächsten Funktionen sind das Ergebnis der Beiträge, Entscheidungen und Auseinandersetzung innerhalb der Community und sie entstehen durch individuelle oder gemeinsame Initiative der beteiligten Entwickler.

Für die beteiligten Unternehmen erwächst daraus das Problem, nicht genau planen zu können, wann die Software über gewünschte funktionsfähige Features verfügen wird. Für die Dienstleister bedeutet dies eine grundsätzliche Ungewissheit, wann den Kunden bestimmte Leistungen angeboten werden könne, für den Distributor stellt sich die Frage, wann die Distribution in bestimmten Umgebungen nutzbar sein wird.

7.4.3.2 Probleme der strategischen Ausrichtung Community-basierter Innovationsprozesse

Eng mit dem ersten Punkt verknüpft, ist für die Unternehmen zudem die mangelnde Steuerbarkeit der strategischen Ausrichtung des Entwicklungsprozesses ein Problem, da auch die Diskussion über technische Lösungen innerhalb der Community weitgehend ergebnisoffen geführt wird. So stellt sich immer die Frage der unmittelbaren Anschlussfähigkeit der Lösung für den je eigenen Bedarf.

Das strategische Interesse des Dienstleisters an der inhaltlichen Entwicklung von „MSB" ist kundengetrieben (auch wenn einzelne Entwickler andere individuelle Interessen haben). Daher ist die Steuerungsproblematik aus Sicht des Dienstleisters kein zentrales Problem, solange er die Arbeiten für die Kunden selbst durchführt. Es ist dann ein Problem, wenn die Community nicht mehr wie erwartet funktioniert, so dass die Erwartungen und Bedarfe der Anwender/Kunden nicht erfüllt werden können. Insofern befindet sich der Dienstleister in einem Dilemma zwischen Community und Kunden und muss beide Ansprüche vermitteln:

> *„Und wenn man mit Communities arbeitet, heißt es immer, du bist irgendwo im Nachteil. […] Immer musst du abwägen, ist der Kunde benachteiligt, ist deine Firma benachteiligt, ist die Community benachteiligt, ist der einzelne Entwickler benachteiligt."* (FS17-IT20-1)

Entscheidungen, die in der Community getroffen werden, sind aus der Perspektive des Dienstleisters oft kompliziert und langwierig:

> *„Die Community-Effekte dahinter sind natürlich schwierig. In Firmen kriegst du das schön gedeckelt, da gibt es Entscheider und die machen die Wege kurz. In Communities sind die Wege lang, die Software ist dann nachher aber durch alle Flamewars durch und die ist veröffentlicht und im Internet, jeder kann seinen Senf dazugeben, die ist in der Regel besser. In der Regel. Aber diese Abstimmungseffekte machen dann eigentlich die Problematik aus."* (FS17-IT20-1)

Auch der Linux-Distributor hat seine eigene Roadmap für seine Produkte und die dafür benötigen Personalressourcen. Seine Produkte haben strategisch gewollte und wichtige Überschneidungen mit „MSB". Andererseits gibt es andere Features von „MSB", in denen er weder Kompetenz noch Interessen hat, dennoch sind auch diese Bestandteil seines Gesamtproduktes. D.h. die eigenen Entwicklungen können zwar weitgehend unabhängig von der Community vorangetrieben werden (s.u.), das eigene Geschäftsmodell funktioniert aber insgesamt besser, wenn die Entwicklung im

Konsens und in Kooperation mit der Community erfolgt. Daher ist auch der Linux-Distributor von den Problemen und Schwierigkeiten im Planungsprozess betroffen.

Vorfälle, wie der oben bereits skizzierte Patt in der Auseinandersetzung um die weitere Entwicklung der Software, der die weitere Entwicklung blockiert, betreffen insofern beide Unternehmen in ihrem Geschäftsmodell, da sie beide auf eine funktionierende Community angewiesen sind, wie es ein Vertreter von UNI1 formuliert:

> *„Wenn das MSB-Team damals wirklich gesagt hätte, wir stellen MSB-Version A ein und machen nur MSB-Version B, dann hätten wir MSB-A geforkt. Hätten gesagt, wir nennen MSB-A anders und führen es alleine weiter, auch mit irren Verlusten an Mensch und Material. Die anderen hätten dann ihr eignes MSB machen können. Aber die Entscheidung ist zum Glück ausgeblieben."* (FS17-IT20-1)

Es gilt insofern für beide Unternehmen, dass Leistungen, die kooperativ innerhalb der Community entwickelt werden, den je eigenen Aufwand für Entwicklungen reduzieren. Daher ist der Versuch, die Community strategisch zu beeinflussen, beiden Unternehmensstrategien inhärent. Wie werden im Folgenden rekonstruieren, wie die Unternehmen dieses Ziel zu erreichen versuchen.

7.4.4 Strategische Lösungsansätze der Unternehmen

In beiden Unternehmen haben wir Ansätze identifizieren können, mit den skizzierten Unwägbarkeiten gemeinschaftlicher Governance umzugehen. Die Ansätze werden im Folgenden zwar übergreifend diskutiert, finden sich jedoch in unterschiedlichem Mischungsverhältnissen in den jeweiligen Unternehmen, auf die in den Unterabschnitten genauer eingegangen wird.

7.4.4.1 Beschäftigung von Mitarbeitern, die Hauptentwickler in der Community sind

Die zentrale Möglichkeit, Einfluss auf die Entwicklung der Software und die Entscheidungen in der Community zu nehmen, besteht darin, führende Köpfe der Community als Mitarbeiter im Unternehmen zu beschäftigen. In einigen Fällen sind die Entwickler bereits in dem Unternehmen beschäftigt, wenn sie beginnen in der Community mitzuarbeiten, nicht selten wechseln wichtige Entwickler aus der Community auch zu einem der beiden Unternehmen, die sich in der Community stark engagieren. Beide Unternehmen betreiben eine strategische Personalpolitik, die nicht nur darauf ausgerichtet ist, eigene Mitarbeiter als Hauptentwickler in der Community aufzubauen, sondern auch gezielt Entwickler aus der Community als Mitarbeiter zu gewinnen und einzustellen. So wurde z.B. in IT20 eine Person eingestellt, die eine zentrale und vermittelnde Rolle innerhalb der Community einnahm:

> *„Und das war zum Beispiel extrem hilfreich, dass [Name des neuen Mitarbeiters – PF] bei uns an Bord kam, weil der [Name eines bestehenden Mitarbeiters – PF] z.B. ein echter MSB-A-Verfechter war und sich auch viele Kämpfe geleistet hat, während [Name des*

neuen Mitarbeiters – PF] ein sehr ruhiger und ausgeglichener Kollege, sagte, ja dann wollen wir mal gucken. Und der hat auch viele MSB-A-Themen entwickelt und der hat einen Großteil der Arbeit gemacht, einfach diesen Kompromiss dann nachher zusammen zu stöpseln, während andere Leute dann die Schminkarbeit übernommen haben und [Name eines anderen Mitarbeiters entfernt – PF] z.B. nachher einen Release backen musste und sagte, naja, das machen wir dann mal. Das ist dann wieder ein Beispiel für richtig erfolgreiche Community-Prozesse." (FS17-IT20-1)

Beide untersuchte Unternehmen haben Mitarbeiter, die gleichzeitig Hauptentwickler in der Community sind. Deren persönliche Beiträge zur Entwicklung der Software in der Community sind die zentrale Form der Beteiligung der Unternehmen. Damit tragen sie der gemeinschaftlichen Governance der Community insofern Rechnung, als sie auf der Arbeitsebene über Entwickler kooperieren, statt direkt als Unternehmen in einen Aushandlungsprozess mit der Community einzutreten. Die persönlichen Beiträge der Entwickler entstehen zu wesentlichen Teilen im Rahmen ihrer Arbeitszeit beim Unternehmen und überwiegend im Auftrag des Unternehmens. D.h. die Entwickler agieren im Unternehmen durchaus im Kontext der unternehmensinternen hierarchischen Strukturen.

Dass dies nicht zu dauerhaften, grundlegenden Widersprüchen führt, ist dadurch zu erklären, dass die Beiträge in der Community auf Ergebnissen ihrer Arbeit im Interesse der Wertschöpfung des Unternehmens Zeitspielräume in ihrer Arbeitszeit nutzen können, die mit Kundenaufträgen finanziert sind. D.h. das Unternehmen beauftragt die Entwickler selbst mit bestimmten Entwicklungsarbeiten. Konkret bereiten sie die Arbeitsergebnisse aus dem Unternehmen so auf, dass sie als Beiträge in der Community sinnvoll sind und akzeptiert werden, sie werden in der Community „upstream gebracht". Dabei haben die Unternehmen ein strategisches Interesse daran, dass die Arbeitsergebnisse der Entwickler im Unternehmen so weit wie möglich „upstream gebracht" werden.

„Es ist nicht so, dass wir uns hinsetzen und einfach entwickeln wozu wir Lust und Laune haben. Das können wir vielleicht zu fünf bis zehn Prozent, kann man sowas mal machen. Aber je nach Auftragslage. Es gab auch schon Zeiten, da war mehr Zeit für sowas, Community-Pflege und einfach generelles „MSB"-Voranbringen. Ist in letzter Zeit nicht so gewesen, da haben wir wirklich fast ausschließlich im Kundenauftrag gearbeitet, aber natürlich immer im Sinne des „MSB"-Teams. Also es macht überhaupt keinen Sinn, für einen Kunden zu arbeiten und die Sachen an Upstream, also an „MSB".org vorbei zu entwickeln. Es gibt manchmal den Fall, dass ein Kunde den expliziten Wunsch hat, dass wir eine Erweiterung machen, die nur für ihn proprietär ist. Ist aber in den seltensten Fällen sinnvoll, weil Upstream sich weiterentwickelt und jedes Mal, wenn der Kunde auf eine neue Upstream-Version will, muss seine private Änderung wieder angepasst und so weiter werden. Langfristig erzeugt das nur Kosten und unnötige Arbeit.

> *Viel besser ist es, Sachen, die generell nützlich sind, direkt Upstream zu bringen. Dann haben zwar auch tendenziell alle anderen was davon, andererseits profitiert auch der Kunde davon, dass die Community das Ding weiter pflegt. Es wird automatisch mitgepflegt, es besteht eine Qualitätssicherung, über die man sich nicht mehr kümmern muss, weil es alle testen, sozusagen. […] Das heißt, wir versuchen das immer so zu machen, wenn es sich irgendwie machen lässt, für unsere Kunden […] aber auch im Sinne von „MSB".org die Upstream-Entwicklung, das irgendwie unter einen Hut zu bringen. Manchmal dauert es ein paar Monate bis Sachen, die für einen Kunden gemacht sind, dann wirklich upstream sind. Aber vielmehr in der Regel auch nicht."* (FS17-IT20-3)

Daher sind die Unternehmen i.d.R. bereit, den Entwicklern (bezahlte) Arbeitszeit einzuräumen, in der sie dies tun. Das grundlegende Prinzip der Arbeitsorganisation beim Dienstleister ist allerdings, dass Kundenaufträge immer vorgehen, d.h. sie werden zunächst abgearbeitet (im Rahmen der Arbeitszeit), erst dann können die Entwickler ihre eigenen Entwicklungen und Projekte auch im Rahmen der betrieblichen Arbeitszeit vorantreiben. Zusätzlich fließt allerdings oft noch ein mehr oder weniger großer Anteil von unbezahlter Arbeit der Entwickler ein. Hier zeigt sich deutlich das Spannungsverhältnis zwischen der hierarchischen Arbeitsorganisation im Unternehmen und den Spielräumen, die die Entwickler haben müssen, um ihre Rolle in der Community spielen zu können. So haben viele der Hauptentwickler auch im Unternehmen eine Position mit (mehr oder weniger) Führungsfunktionen, die ihnen die Abstimmung der unterschiedlichen Anforderungen insofern ermöglicht, als sie in beiden Rollen Entscheidungen selbst verantworten können. D.h. sie können die Arbeit im Unternehmen in einem gewissen Rahmen so organisieren, dass sie hierbei bereits die Interessen der Community und deren interne Prozesse mitdenken und mitberücksichtigen – und umgekehrt. Dies hat im Rahmen der Unternehmenshierarchie natürlich Grenzen: Im Zweifel ist klar, dass für die Arbeit, die im Unternehmen und im Rahmen der Arbeitszeit geleistet wird, die Interessen des Unternehmens Vorrang haben, d.h. Aufträge im Unternehmen werden zuerst abgearbeitet. Wenn hier keine Zeit bleibt für Aufgaben in der Community, bleibt den Entwicklern nur die Verlagerung dieser Tätigkeiten in die Freizeit. D.h. das konkrete Spannungsverhältnis (er)trägt letztlich der Entwickler oder die Anforderungen aus der Community werden auf einen späteren Zeitpunkt verschoben, was zu den bereits erwähnten Unwägbarkeiten im Entwicklungsprozess führt.

Die Mitarbeit von Angestellten der Unternehmen als Hauptentwickler in der Community führt also dazu, dass diese immer in einer Doppelrolle mit zwei unterschiedlichen Orientierungslogiken agieren. Sie sind zum einen den Weisungen der Unternehmen unterworfen, zum anderen sind sie als Teil der Community auch an die dort geltenden Normen und Organisationsformen gebunden. Und schließlich haben sie als Entwickler auch eigene inhaltliche und berufliche Interessen.

Diese doppelten (oder sogar dreifachen) Orientierungsrahmen sind den Beteiligten präsent und müssen im alltäglichen Handeln ausbalanciert werden. Dies gilt für

das Unternehmen genauso wie für die Entwickler selbst. Der Geschäftsführer von IT20 beschreibt diese Konstellation folgendermaßen:

> *„Der einzelne Entwickler hat eine bestimmte Motivation, das MSB-Team hat ein Ziel und das Unternehmen, für das der Entwickler arbeitet, hat auch eine Strategie. Drei unterschiedliche Orientierungen. Und es wäre schön, wenn das parallel ist – ist es natürlich nie […] Oder wenigstens, wenn es ungefähr in die gleiche Richtung geht. Aber manchmal geht es völlig auseinander und dann hat der einzelne Teilnehmer in dieser Community ein Problem. Und dann gibt es Leute wie mich, ich arbeite seit […] Jahren mit dieser Community. Ich bin kein Teil des Entwicklerteams, ich habe ein Firmeninteresse. Ich sehe die Team-Interessen, ich sehe auch die verschiedensten Entwickler-Interessen, aber ich sehe auch noch das Interesse meines Kunden. Also ein bunter Strauß an Interessen."* (FS17-IT20-1)

Als Mitglieder der Community bringen die Entwickler die Beiträge zur Weiterentwicklung der „MSB"-Software unter ihrem eigenen Namen ein und verantworten sie auch selbst:

> *„Wie gesagt, die Patches die wir machen, als Entwickler […], die gehen sowieso alle irgendwie mal upstream (in die „MSB"-Software) oder an den Kunden. Und zwar unter persönlichem Copyright. IT20 an sich verteilt „MSB" ja nicht im Sinne eines GPL-Distributors. Sondern wir machen Patches, die wir an Kunden ausliefern und die wir upstream einbringen."* (FS17-IT20-2)

Die Mitarbeit in der Community setzt in jedem Fall die Bereitschaft der Entwickler voraus, sich zusätzlich einzubringen und dies ggf. auch über die bezahlte Arbeitszeit hinaus in der Freizeit zu tun. Diese Bereitschaft erwächst aus der Zugehörigkeit zur Community, dem gemeinsamen Ziel und auch aus der sozialen Position und Anerkennung, die der Entwickler bzw. die Entwicklerin in der Community erfährt. Auch den Entwicklern ist diese Doppelrolle durchaus präsent, wie folgende Aussage zeigt:

> *„Sagen wir mal so, ich habe immer quasi meinen „MSB"-Team- und meinen IT20-Hut auf. Und auf den Mailinglisten poste ich immer @IT20.de. Das heißt, das ist völlig klar, dass ich von IT20 komme und wenn ich dann irgendwelche schlauen Kommentare abgebe. Gut, wir haben ja auch laut deutschem Gesetz immer unseren Email-Footer, Geschäftsführer, Handelsregister und Telefonnummer, und ich hoffe schon so ein bisschen darauf, dass die Leute, wenn sie nicht weiterkommen, dann irgendwann mal bei uns anrufen. Das ist die Hoffnung. […] Es passiert ganz selten, vielleicht ein Mal im Jahr, dass ich jemanden, der in der Community-Mailingliste eine Frage stellt, tatsächlich direkt anmaile und sage, hier, ich habe jetzt keine Zeit mehr, mich kostenfrei drum zu kümmern, soll ich meinen Vertrieb auf dich loslassen? Das passiert auch, ist aber selten, und natürlich mit unterschiedlichem Erfolg. Klar, das ist sicherlich ein Zielkonflikt. […] Aber wenn jemand quasi gute Patches liefert, die das Projekt weiterbringen, dann hat IT20 letztendlich auch was davon. Dass „MSB" halt weiter benutzt wird und wenn „MSB" tot ist, ist die MSB-Abteilung von IT20 auch tot. Also das ist immer schwierig zu fassen."* (FS17-IT20-2)

Dabei sind der unterschiedliche soziale Kontext und die daraus folgenden Interessenlagen sehr präsent, wie folgende Aussage eines Entwicklers zeigt:

> *„F: Gibt es Regeln, die dazu führen würden, dass, wenn IT20 sich dagegen verhält, zum Beispiel deine Position oder deine Beziehung zu den Personen im MSB-Team gefährdet würde? Also gibt es irgendwie Rücksichtnahmen?*
>
> *Regeln, nein. Die einzige echte Regel die wir haben, ist die GPL. IT20 sollte keine GPL verletzten. Rücksichtnahmen, ja klar. Das gibt es. Dass ich in meiner Rolle einerseits als IT20-Angestellter, andererseits als MSB-Team-Mitglied – dass da hin und wieder mal irgendwelche Interessen nicht ganz kongruent sind, das gibt es definitiv. Und da muss ich dann im Einzelfall gucken muss, wie verhalte ich mich. Ich meine, es ist halt auch immer IT20s ureigenstes Interesse, irgendwie mit MSB Geld zu verdienen. Aus der Software an sich, GPL, kann man kein Geld verdienen, das heißt wir versuchen als IT20 immer wieder, Geschäftsmodelle rund um MSB zu finden. Und da muss man halt im Einzelfall gucken, würde das die Community toll finden, und wenn sie es nicht toll findet, wie groß ist der Schmerz der Community damit? Können wir es uns noch erlauben als IT20? Oder würde das halt irgendwie sehr böse aufstoßen? Muss man im Einzelfall immer gucken. Aber klar, gibt es, natürlich."* (FS17-IT20-1)

Der Gesprächspartner von IT20 formuliert hier sehr deutlich, dass der Umgang mit community-basierter Governance ein ständiges Ausbalancieren und Reagieren auf die anderen Akteure voraussetzt. Und dass dieser Umgang nicht primär wirtschaftlicher Logik folgen kann, sondern immer die unterschiedlichen – meist nicht wirtschaftlich-rationalen Orientierungen der übrigen Akteure erahnen und in Rechnung stellen muss. Seine direkten Steuerungsmöglichkeiten sind auf das Unternehmen begrenzt und auch hier ist gegenüber den Hauptentwicklern Zurückhaltung angesagt. Der zentrale Mechanismus läuft über die Kontrolle der Ressourcen und natürlich über Vorschläge und Überzeugungskraft.

> *„Also so eine Community ist ein bunter Strauß an Interessen. Das zeichnet ja, das ist ja das, was du da als Überschrift über diese ganze Community schreiben kannst, Multi-Stakeholder, klassisches Multi-Stakeholder-Prinzip. Das heißt, unterschiedlichste Interessenvertreter mit unterschiedlichen Einflusssphären und Machtnutzungsmöglichkeiten. […] Und es gibt bestimmte Faktoren, wie Geld, viel Geld macht unglaublich viel platt. Weil viel Geld auf einem Projekt bedeutet für uns z.B., dass die Entscheidungen alternativlos sind, weil es dann nämlich heißt, willst du das Projekt haben oder nicht? […] Und es gibt dann jede Menge Loyalitäten, die sind personell immer incentiviert oder finanziell oder sonst wie. Und in dieser Gemengelage bewegt man sich."* (FS17-IT20-1)

7.4.4.2 „Rallying Calls“

Um auf den Entwicklungsprozess der Software in der Community Einfluss zu nehmen, beteiligen sich die Unternehmen in vielfältiger Weise an den Meinungsbildungsprozessen in der Community. Also nicht nur indirekt über die bei ihnen beschäftigten Hauptentwickler, sondern auch durch die Organisation von Konferenzen, Meetings, Workshops zu denen sie die Entwickler aus der Community einladen, auf denen die Entwicklungsperspektiven diskutiert und an gemeinsamen Projekten gearbeitet wird. Selbst Microsoft bietet seit einigen Jahren sogenannte Blogfests und Workshops für die Entwickler an. In diesen Veranstaltungen bringen die Unternehmen Vorschläge für die Weiterentwicklung ein, um andere Entwickler für die Mitarbeit und Unterstützung zu gewinnen.

Die jährlich stattfindenden Entwicklerkonferenzen der Community sind eine wichtige Einflussmöglichkeit der Unternehmen auf die Community, auf die wir in unserer Studie gestoßen sind. Diese Konferenzen sind zum einen wichtig für den direkten und persönlichen Kontakt der Entwickler untereinander, zum anderen finden auf diesen Konferenzen auch wichtige Diskussionen über die weitere strategische Ausrichtung des Projekts statt.

Die Strategie der Unternehmen in Bezug auf die Konferenz kann unter dem Begriff der „Rallying Calls“ gefasst werden. Auf dem als Konferenz organisierten Entwicklertreffen gibt es – ganz ähnlich wissenschaftlichen Veranstaltungen – Fachvorträge unterschiedlicher Akteure, auf denen zukünftige technische Probleme und mögliche Lösungsstrategien diskutiert werden. Vortragende sind sowohl Entwickler der Community, aber auch VertreterInnen von Anwenderunternehmen, beteiligten Unternehmen und in den letzten Jahren sogar Microsoft selbst. Das Vorhaben aller Akteure besteht darin, die eigenen Interessen an der weiteren Entwicklung zu präsentieren und möglichst Mitstreiter für die eigenen Projekte und Entwicklungsideen zu gewinnen. Auch auf diese Weise können sich die beteiligten Unternehmen einen Einfluss auf den Fortgang der Entwicklung verschaffen und versuchen, die zukünftige Entwicklung in die gewünschte Richtung zu lenken.

7.4.4.3 Alleingang als Fallback-Lösung

Der dritte Ansatz, den Unternehmen zur strategischen Einflussnahme auf den Entwicklungsprozess verfolgen, ist gleichzeitig der defensivste und lässt sich unter dem Stichwort „Alleingang“ fassen. Die oben diskutierten Ansätze, die Entwicklung in die gewünschte Richtung zu lenken, sind zweifelsfrei wichtig in beiden untersuchten Fällen. Aufgrund der skizzierten Unsicherheiten in der Planung und der strategischen Steuerung der Entwicklung, sind jedoch beide Ansätze mit verbleibenden Unsicherheiten für die beteiligten Unternehmen verbunden. In vielen Fällen reagieren beide Unternehmen daher in dringenden und akuten Fällen damit, dass sie wichtige Entwicklungen in Eigenregie betreiben und erst anschließend in die Community zurückspeisen:

> *„Man kann sich auf die Community nicht verlassen. IT20 kann keine Zusage treffen, dass irgendjemand anders in der Community in irgendeinem Zeitrahmen irgendetwas tut. Das tut IT20 nicht. Also entweder wir können das wirklich selber und können ein Angebot abgeben, oder IT20 sagt, das kostet ein halbes Jahr Arbeit, und der Kunde sagt, nein das ist mir zu teuer. Wir können einem Kunden sicherlich sagen, sprich mal den an, vielleicht hat der eine Idee, vielleicht hat der ja schon Patches, irgendwas. Aber IT20 kann ja keine Community-Leistung verkaufen. Sondern wir können nur unsere Leistung verkaufen. Und dass es etwas im MSB-Umfeld gibt, wo wir keine Kompetenz hätten und das rausgeben müssten, das ist noch nicht passiert."* (FS17-IT20-1)

Die eigenständige Weiterentwicklung der OSS erfolgt beim Entwicklungsdienstleister in der Regel aufgrund von Kundenanforderungen. Allerdings betreibt IT20 auch gezielte Akquise von Kundenaufträgen in dem Sinne, dass sie für bestimmte Entwicklungsaufgaben für die Open Source Software aktiv versuchen Kundenaufträge zu generieren. Dies ist in vielen Fällen erfolgreich, hat aber Auswirkungen darauf, was für „MSB" entwickelt wird. Der Dienstleister kann aufgrund seines Geschäftsmodells nur eingeschränkt eine unabhängige Roadmap für die Weiterentwicklung von „MSB" verfolgen.

Für den Linux-Distributor stellt sich das Problem auf andere Weise. Aufgrund des Geschäftsmodells ist IT19 nicht auf Kundenaufträge angewiesen, sondern betreibt eine eigenständige Produktstrategie. Es gibt eigene Entwicklungsprojekte, die zwar an die OSS der Community anknüpfen, aber in ihrer Funktionalität deutlich darüber hinausgehen und auf Alleinstellungsmerkmale ihrer Linux Distribution ausgerichtet sind. Die für das eigene Produkt wichtigen Funktionen der Open Source Software werden von IT19 in Eigenregie entwickelt und erst anschließend in die offene Codebasis integriert. Allerdings findet die eigene Entwicklung in den von uns untersuchten Beispielen meist in enger und offener Kollaboration mit anderen Entwicklern aus der Community statt. Auch aufgrund der Größe des Unternehmens verfügt IT19 über größere finanzielle Reserven als das bei IT20 der Fall ist. Bei dieser Strategie besteht das Risiko vor allem darin, dass die in Eigenregie entwickelten Softwaremodule nicht in das OSS-Produkt übernommen werden.

7.5 Fazit und Ausblick

Die hier vorgestellt Fallstudie hatte zum Ziel, den Zugriff von Unternehmen auf externes Wissen in einem verteilten Innovationsprozess zu analysieren, der im Rahmen einer OSS-Community koordiniert wird. Im Fokus standen dabei der konkrete Austausch von Wissen zwischen Unternehmen und Akteuren der Community im Entwicklungsprozess und der Umgang der Akteure mit dem Spannungsverhältnis, das aus den unterschiedlichen Logiken und Handlungsorientierungen von hierarchischer und gemeinschaftlicher Governance erwächst. Wie eingangs herausgearbeitet, lässt sich Wissenstransfer als komplexes Problem der Neugenerierung von Wissen

im Rahmen der gemeinsamen sozialen Praxis beschreiben. Wie betont wurde, beinhaltet eine solche Neugenerierung aus verteilten Wissensbeständen (auch als Rekontextualisierung bezeichnet) mehr als das Teilen expliziter Wissensbestandteile. Vielmehr erfordert eine erfolgreiche Rekontextualisierung von Wissen auch den Zugang zu impliziten, oft erfahrungsbasierten Formen von Wissen, die in einer gemeinsamen Praxis geteilt werden können.

Die spezifische Frage, die anhand der Fallstudie der Kollaboration zwischen Unternehmen und einer OSS-Community beantwortet werden sollte, war also die Frage, welche spezifische Praxis des Wissenstransfers die OSS-Community ermöglicht, welche (Steuerungs-)Probleme daraus für die beteiligten Unternehmen resultieren und welche Strategien zur Lösung die Unternehmen entwickeln.

Die Untersuchung konnte rekonstruieren, dass die Community wesentliche Eigenschaften aufweist, die einen erfolgreichen Transfer nicht nur expliziten, sondern auch impliziten Wissens fördern. Neben der technischen Infrastruktur, die für OSS-Communities typisch ist, waren es vor allem die für Gemeinschaften typischen sozialen Mechanismen einer gemeinsamen kollektiven Identität und Zielsetzung, der „Institutionalisierung des Kollektiven" (Dolata & Schrape 2014) und der internen Differenzierung und Organisierung, die für die soziale Praxis innerhalb der Community ausschlaggebend waren. Wie sich im weiteren Verlauf der Untersuchung zeigte, waren dieselben Merkmale von Gemeinschaften, die den Wissenstransfer positiv unterstützten, gleichzeitig auch die Quelle von Steuerungsproblemen für die beteiligten Unternehmen. Gemeinschaften beruhen auf selbstorganisierter und freiwilliger Kooperation und stehen dadurch in Konflikt mit der hierarchischen und an den eigenen Unternehmenszielen orientierten internen Governance der beteiligten Unternehmen. Für die Unternehmen bedeutet dies, dass sie bei der Verfolgung ihrer eigenen Ziele an die oftmals langwierigen und komplizierten Aushandlungsprozesse in der Community gebunden sind, was die zeitliche und strategische Steuerung des Entwicklungsprozesses erschwert.

Unsere Ergebnisse bestätigen damit z.T. die Ergebnisse von Dobusch & Quack (2011) insofern, als dass auch wir zu dem Schluss kommen, dass Organisationen die Gemeinschaften, auf denen sie aufbauen, nur sehr unzureichend steuern können:

> *„In beiden Fällen zeigt die Analyse, dass Organisationen die Entwicklung der relevanten Gemeinschaften von Beitragenden nicht im engeren Sinne steuern, sondern diese nur indirekt durch das Management der Grenzbeziehungen beeinflussen können."* (Dobusch & Quack 2011)

Allerdings zeigen unsere Ergebnisse im Gegensatz zu denen von Dobusch & Quack, dass es Unternehmen durchaus gelingen kann, über das reine Management der Grenzbeziehungen hinaus strategischen Einfluss auf die Communities zu nehmen. Mit strategischer Personalrekrutierung, Rallying Calls innerhalb der Community und der Fallback-Option des Alleingangs wurden drei Ansätze diskutiert, auf die Unternehmen in unserem Fall zurückgreifen. Bei aller Unsicherheit, die diese Ansätze nach wie vor beinhalten, stellen sie doch Möglichkeiten für die Unternehmen bereit,

auf den Entwicklungsprozess und die strategische Ausrichtung der Gemeinschaft Einfluss zu nehmen. Der Grund für diesen unterschiedlichen Befund liegt aus unserer Sicht in dem Unterschied zwischen nicht-kommerziellen und kommerziellen Organisationen (Unternehmen) und dem besonderen Mix der Governance-Formen begründet. Die von Dobusch & Quack untersuchten Organisationen sind gewissermaßen die formalen Organisationen der dahinterstehenden Gemeinschaften und versuchen, deren Anliegen mit anderen Akteuren im Feld (anderen Organisationen) zu vermitteln. In unserem Fall handelt es sich jedoch um ökonomische Akteure, die eigene, klar von denen der Community zu unterscheidende, wirtschaftliche Interessen verfolgen und die intern hierarchisch organisiert sind. Die Unternehmen stellen wichtige Personalressourcen für die Community zur Verfügung, über die sie durch das Arbeitsverhältnis (wie auch immer eingeschränkte) hierarchische Weisungs- und Kontrollmöglichkeiten haben. Dies löst – wie wir zeigen konnten – die Steuerungsproblematik nicht, aber es befördert einen stetigen Aushandlungsprozess zwischen der Community und den Unternehmen, in dem letztere ihren Einfluss geltend machen können. Kennzeichnend für den Typ der Community-geführten OSS-Projekte ist allerdings, dass die Unternehmen nicht direkt in die internen Entscheidungsprozesse und Strukturen der Community steuernd eingreifen können. Der besondere Mix der Governance-Formen beruht hier darauf, dass die jeweiligen internen Governancemechanismen der Community auf der einen Seite und des Unternehmens auf der anderen Seite unabhängig voneinander gültig bleiben, sie sind gewissermaßen komplementär.

Wir haben es hier also mit einem Mix aus gemeinschaftlicher und hierarchischer Governance zu tun, bei dem die unterschiedlichen Governancemechanismen unterschiedliche Funktionen erfüllen und so auch von den Akteuren (strategisch) eingesetzt werden. Die Open Source Community als intern gemeinschaftlich organisierte Governance mit entsprechenden Koordinationsmechanismen wird für den Austausch und die Integration von Wissen und Entwicklern (als Person und Wissensträger) im Kontext des gemeinschaftlich organisierten Entwicklungsprozesses genutzt. Das eigenständige institutionelle Setting der Community bietet einen überbetrieblichen Rahmen für die Integration der Beiträge aus den verschiedenen involvierten Unternehmen. Die Community koordiniert auf überbetrieblicher Ebene die verteilte Innovation, die offen ist für heterogene Akteure und verbreitet das Produkt als kollektives Gut (ähnlich argumentieren Gulati, Puranam & Tushman 2012 in Bezug auf die Herausbildung von Metaorganisationen für die Koordination von verteilten Innovationen).

Im Gegenzug bietet die strategische Beteiligung der Unternehmen der Community den Zugang zu Personalressourcen, über die die Community selbst nicht verfügt. Der volatile, gemeinschaftliche Innovationsprozess kann durch diese hierarchisch gesteuerten Personalressourcen der Unternehmen nachhaltig gesichert und stabilisiert werden. Allerdings erzeugt der Zugriff auf Personalressourcen der Unternehmen gleichzeitig auch Risiken für die unabhängige Entwicklung der Open Source und eine spezifische Abhängigkeit vom Unternehmen.

Die Unternehmen sind intern hierarchisch organisiert, mit Arbeitsverhältnissen und innerbetrieblichen Planungs- und Steuerungsprozessen, denen auch die Entwickler unterliegen. Allerdings bringt die Externalisierung der Koordination des verteilten Innovationsprozesses an die Community Steuerungsprobleme für die Unternehmen und eine spezifische Abhängigkeit von der Open Source Community mit sich.

Die Entwickler finden sich gewissermaßen an der Schnittfläche der beiden Governance-Formen. Sie agieren in dem Spannungsfeld zwischen beiden Handlungsorientierungen. Ihre Doppelrolle ist ein Schlüssel für das Verständnis dieses spezifischen Mix aus gemeinschaftlicher und hierarchischer Governance.

Mit der Untersuchung eines Falles beruhen unsere Ergebnisse selbstverständlich auf einem begrenzten empirischen Material. Dies ist vor allem vor dem Hintergrund der Varianz von OSS-Communities kritisch zu reflektieren. Wie Schrape (2015) herausarbeitet, gibt es eine breite Varianz an Communities, die sich besonders im Hinblick auf die in ihnen wirksamen Koordinationsmuster (hierarchisch/horizontal) und ihre Abhängigkeit von etablierten Unternehmen voneinander unterscheiden. Unsere Ergebnisse zeigen in dieser Hinsicht, dass aus einer Nähe zu Unternehmen und dem Umstand, dass ein Großteil der Entwicklung der Gemeinschaft von Unternehmen direkt oder indirekt (über die bereitgestellten Entwickler) finanziert wird, nicht direkt auf hierarchischen Einfluss geschlussfolgert werden darf. Die Institutionen und Praxen der Community können auch in diesem Fall eine eigenständige Ordnung darstellen, die der hierarchischen Governance innerhalb der Unternehmen Grenzen setzt. Allerdings ist zu erwarten, dass dieser Konflikt zwischen der eher horizontalen (gemeinschaftlichen) und der hierarchischen Governance (innerhalb der Unternehmen) umso stärker ist, je stärker Communities eine eigenständige gemeinschaftliche Institutionalisierung durchlaufen haben. Unsere Ergebnisse legen den Schluss nahe, dass, je stärker die Strukturen und Prozesse der Communities in eigenständiger Weise verfasst sind, desto größer werden die Konflikte zwischen der internen hierarchischen Governance der Unternehmen und der Governance der Communities sein und desto größere Anpassungen werden von den beteiligten Unternehmen erzwungen werden.

Unsere Ergebnisse verweisen damit auf ein interessantes Spannungsfeld zwischen zwei verschiedenen Ordnungsmustern in kollaborativen Innovationsprojekten. Auch wenn OSS-Communities mittlerweile in der proprietären Softwareindustrie zur normalen Praxis gehören und nicht mehr als Vorbote revolutionärer Umwälzungen des Modus der Softwareproduktion angesehen werden können, wie Schrape (2015) zu Recht hervorhebt, ist die Einbindung von Community-basierter Wissensproduktion in kommerzielle Wertschöpfungsprozesse damit keineswegs trivialisiert worden, sondern beinhaltet nach wie vor z.T. erhebliche Herausforderungen für die beteiligten Unternehmen. Und auch wenn OSS und proprietäre Softwareproduktion nicht notwendigerweise als sich ausschließende Arten der Softwareproduktion angesehen werden müssen, erfordert die erfolgreiche Integration doch durchaus orga-

nisatorische und personalpolitische Anpassungsleistungen der Unternehmen. Unsere Ergebnisse verweisen damit auf weiteren Forschungsbedarf an dieser Schnittstelle zwischen Community-basierter und proprietärer Softwareentwicklung.

7.6 Literatur

Berger, P. L. & Luckmann, T. (1984): *Die gesellschaftliche Konstruktion der Wirklichkeit*. Frankfurt a.M.: Fischer Taschenbuch Verlag.

Dahlander, L. & Magnusson, M. G. (2005): Relationships between open source software companies and communities: Observations from Nordic firms. In: *Research Policy,* 34(4), S. 481–493.

David, P. A. & Shapiro, J. S. (2008): Community-based production of open-source software: What do we know about the developers who participate? In: *Information Economics and Policy,* 20(4), S. 364–398.

Dobusch, L. & Quack, S. (2011): Interorganisationale Netzwerke und digitale Gemeinschaften: Von Beiträgen zu Beteiligung? In: *Managementforschung,* 21, S. 171–213.

Dolata, U. & Schrape, J.-F. (2014): Kollektives Handeln im Internet. Eine akteurtheoretische Fundierung. In: *Berliner Journal für Soziologie,* 24(1), S. 5–30.

Giuri, P., Rullani, F. & Torrisi, S. (2008): Explaining leadership in virtual teams: The case of open source software. In: *Information Economics and Policy,* 20 (4), S. 305–315.

Gläser, J. (2006): *Wissenschaftliche Produktionsgemeinschaften die soziale Ordnung der Forschung*. Frankfurt am Main [u.a.]: Campus-Verlag.

Gläser, J. (2007): Gemeinschaft. In: A. Benz, S. Lütz, U. Schimank & G. Simonis (Hrsg.): *Handbuch Governance – theoretische Grundlagen und empirische Anwendungsfelder*. Wiesbaden: VS Verlag, S. 82–92.

Gulati, R. Puranam, P. & Tushman, M. (2012): Meta-organization design: rethinking design in interorganizational and community contexts. In: *Strategic Management Journal* 33, S. 571–586.

Heidenreich, M. (1995): *Informatisierung und Kultur – Die Einführung und Nutzung von Informationssystemen in italienischen, französischen und westdeutschen Unternehmen*. Wiesbaden: VS Verlag.

Heidenreich, M. (2003): Die Debatte um die Wissensgesellschaft. In S. Böschen & I. Schulz-Schaeffer (Hrsg.): In: *Wissenschaft in der Wissensgesellschaft*. Opladen: Westdeutscher Verlag, S. 25–55.

Hollingsworth, J. R. (2000): Doing institutional analysis: implications for the study of innovations. In: *Review of International Political Economy,* 7(4), S. 595–644.

Hollingsworth, J. R. & Boyer, R. (1997): Coordination of economic actors and social systems of production. In J. R. Hollingsworth & R. Boyer (Hrsg.): *Contemporary capitalism – the embeddedness of institutions*. Cambridge: Cambridge University Press, S. 1–47.

Krogh, G. v. (2011): Knowledge Sharing in Organizations: The Role of Communities. In: M. Easterby-Smith & M. A. Lyles, (Hrsg.): *Handbook of Organizational Learning & Knowledge Management*. West Sussex: Wiley, S. 403–433.

Lange, M. (2009): *Interoperability and Microsoft: Then and Now*. SSRN eLibrary.

Mateos-Garcia, J. & Steinmueller, W. E. (2008): The institutions of open source software: Examining the Debian community. In: *Information Economics and Policy,* 20(4), S. 333–344.

O'Mahony, S. (2007): The governance of open source initiatives: what does it mean to be community managed? In: *Journal of Management & Governance,* 11(2), S. 139–150.

O'Mahony, S. & Lakhani, K. R. (2011): Organizations in the Shadow of Communities. In: *Harvard Business School Working Paper,* No. 11–131.

Polanyi, M. (1985): *Implizites Wissen*. Frankfurt a.M.: Suhrkamp.

Schrape, J.-F. (2015): Open Source Softwareprojekte zwischen Passion und Kalkül. *SOI Discussion Paper 2015–02*. Stuttgart: Universität Stuttgart.

Streeck, W. & Schmitter, P. C. (1985): Community, Market, state – and associations? The prospective contribution of interest governance to social order. In: W. Streeck & P.C. Schmitter (Hrsg.): *Private Interest Government – Beyond Market and State*. London: Sage, S. 1–29.

Tsoukas, H. (1996): The firm as a distributed knowledge system: a costructionist approach. In: *Strategic Management Journal,* 17 (Winter Special Issue), S. 11–25.

Von Hippel, E. (2005): *Democratizing Innovation.* Cambridge, Massachusetts: The MIT Press.

Wenger, E. (2000): Communities of Practice and social learning systems. In: *Organization,* 7(2), S. 225–246.

West, J. & Gallagher, S. (2006): Challenges of open innovation: the paradox of firm investment in open-source software. In: *R&D Management,* 36(3), S. 319–331.

West, J. & Lakhani, K. R. (2008): Getting Clear About Communities in Open Innovation. In: *Industry and Innovation,* 15(2), S. 223–231.

West, J. & O'Mahony, S. (2005): Contrasting Community Building in Sponsored and Community Founded Open Source Projects. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on, 03–06 Jan. 2005* (S. 196c).

West, J. & O'Mahony, S. (2008): The Role of Participation Architecture in Growing Sponsored Open Source Communities. In: *Industry and Innovation,* 15(2), S. 145–168.

Wiesenthal, H. (2005): Markt, Organisation und Gemeinschaft als „zweitbeste" Verfahren sozialer Koordination. In W. Jäger & U. Schimank (Hrsg.) In: *Organisationsgesellschaft*: Wiesbaden: VS Verlag, S. 223–264.

Windeler, A. (2012): Kooperation und Konkurrenz in Netzwerken. Theoretische Uberlegungen zur Analyse des Strukturwandels der Arbeitsorganisation. In: C. Schilcher, M. Will-Zocholl & M. Ziegler (Hrsg.): *Vertrauen und Kooperation in der Arbeitswelt*. Wiesbaden: VS Verlag, S. 23–50.

Bibliographische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet über <http://dnb.dnb.de> abrufbar.

Herausgeberkontakt
Prof. Dr. Jürgen Kädtler
Soziologisches Forschungsinstitut Göttingen (SOFI) an der
Georg-August-Universität Göttingen
Friedländer Weg 31
37085 Göttingen
http://www.sofi-goettingen.de/

Satz und Layout: Lukas Thamm
Umschlaggestaltung: Jutta Pabst
Titelabbildung: sdecoret/Shutterstock.com

https://univerlag.uni-goettingen.de
ISBN: 978-3-86395-347-8
DOI: https://doi.org/10.17875/gup2018-1080

Inhaltsverzeichnis