Ruhr-Universität Bochum
Fakultät für Sozialwissenschaften
WS 12/13; SS 13
Lehrforschungsmodul Kommunikation 2.0
Dozent: Dr. Mario Paul

Freiwillig – Digital – Modern

Eine qualitative Studie über die Entwicklung von Freier und Open Source Software

Behl, Teresa Niklas, Jennifer Sepan, Kathrin

Inhaltsverzeichnis

1. Einleitung

Was den Menschen vom Tier unterscheidet, ist eine seit langem geführte philosophische Debatte (Newen & Bartels 2011). Tomasello vertritt in seinem Buch „Warum wir kooperieren" die These, dass der Mensch sich aufgrund seiner ultra-kooperativen Tendenzen vom Tier abhebt und unterstützt seinen Standpunkt durch eine Vielzahl von interessanten Studien sowohl zum Verhalten von Kleinkindern als auch von Schimpansen (Tomasello 2010). Seiner Ansicht nach ist die Kooperationsfähigkeit einer der entscheidenden Faktoren, die dem Menschen seine einzigartigen kulturellen Leistungen ermöglicht haben. Eine der jüngsten kulturellen Leistungen des Menschen ist sicherlich das Internet mit all seinen technischen Möglichkeiten. Es hat einen neuartigen virtuellen öffentlichen Raum geschaffen, der immer stärkere Auswirkungen auf unser Leben im Alltag hat. Wenn wir nun Tomasellos These ernst nehmen, dann wäre der Mensch nicht Mensch, wenn er nicht ebenfalls in der digitalen Welt neue Formen des Mit- und Füreinander finden würde. Der Frage, wie diese neuen Kooperationsformen entstehen, soll diese Arbeit anhand mehrerer konkreter Fälle mit den Mitteln der qualitativen Sozialforschung nachgehen.

Freiwilliges Engagement vollzieht sich in einem öffentlichen sozialen Raum. Zu diesen zählt seit einigen Jahren ebenfalls das Internet mit seinen verschiedenen sozialen Netzwerken. Darüber hinaus hat sich eine Open-Bewegung entwickelt, die aus einer Vielzahl von engagierten Einzelpersonen und organisierten Gruppen besteht, deren Ziel es ist, anderen freien Zugang zu Wissen, Computerprogrammen und Betriebssystemen zu verschaffen. Wikipedia – ein sogenanntes Open Content Projekt – kennt in Deutschland fast jeder Computernutzer. Es ist das größte Archiv für freie Inhalte und ein hervorragendes Medium, um sich schnell einen Überblick über ein Thema zu verschaffen. Oft wird argumentiert, dass die freie Enzyklopädie Wikipedia im Gegensatz zu kostenpflichtigen Enzyklopädien wie dem Brockhaus-Lexikon nicht ausreichend vertrauenswürdig ist. Schließlich werden die Artikel nicht durch Fachpersonal, sondern durch Freiwillige verfasst. Dass dieses Projekt jedoch einen großen Beitrag für die Gesellschaft leistet, zeigt sich unter anderem im Stern-Enzyklopädien-Vergleich, bei dem Artikel aus beiden Enzyklopädien nach den Kriterien Richtigkeit, Vollständigkeit, Aktualität und Verständlichkeit bewertet wurden. Wikipedia schnitt im Mittel eine Note besser ab als die Artikel der Online-

Ausgabe des Brockhaus. Die Stärken von Wikipedia lagen vor allem im Bereich Aktualität, nur bei der Verständlichkeit wurde der Brockhaus etwas besser benotet. Betrachtet man die Umfrage „Monitor Engagement“ vom Bundesministerium für Familie, Senioren, Frauen und Jugend (2010) fällt auf, dass der Computer, bzw. das Internet im Bezug auf das freiwillige Engagement bisher wenig beachtet wird und meist nur als Freizeitbeschäftigung oder als Medium zur Organisation der Verwaltung und Recherche gesehen wird.

Freie Meinung, freie Bildung und freie Inhalte werden mittlerweile als wichtiger Beitrag für unsere demokratische Gesellschaft gewürdigt. Wieso dann nicht auch freie Software? Freie Software (FS) unterstützt den ideologischen Gedanken der Computernutzer die Kontrolle über ihre Rechner, die teilweise von Firmen, Staaten und Entwicklern ausgeübt wird, zurückzugewinnen. Eine eher praktische Motivation vertreten die Anhänger der Open Source Software (OSS). Ihnen geht es primär um das Schreiben einer effizienten, sich schnell entwickelnden, flexiblen Software unabhängig von Ideologien. Um allen gerecht zu werden, wurde der Begriff “Free/Libre Open Source Software” (FLOSS) eingeführt, der sowohl auf die Freiheiten als auch auf den offenen Quellcode Bezug nimmt. Die Basis dieser Software sind der Quellcode und die zahlreiche Beiträge der Entwickler, die größtenteils ohne finanziellen Eigennutzen, gemeinsam, kreativ und effektiv Projekte ermöglichen, die durchaus mit proprietärer Software konkurrieren können.

Die vorliegende Studie untersucht die Entwicklung[1] von FLOSS-Software. Sie möchte die dort zugrundeliegende sozio-kulturelle Praxis analysieren und versuchen, Einblicke in die Auslöser, Motive und Motivationen dieser Praxis zu gewinnen. Das methodische Vorgehen der Arbeit basiert auf den Grundlagen der Grounded Theory (Strauss & Corbin 1996). Die Auswahl der Interviewpartner erfolgte nach der Theoretical Sampling-Methode (Strauss & Corbin 1996: 148ff). Die Analyse der narrativen Interviews wurde nach mehreren Prinzipien des methodisch kontrollierten Fremdverstehens (Przyborski & Wohlrab-Sahr 2009: 28ff) durchgeführt.

Im ersten Teil werden die geschichtliche Entwicklung, die Struktur und die Philosophie der FLOSS-Gemeinschaft ausführlich erläutert. Im zweiten Teil der Arbeit werden die

[1] Unter dem Begriff Entwicklung ist hier sowohl das konkrete Entwickeln der Software, als auch das zur Verfügung Stellen, also das Veröffentlichen, der Software gemeint, da die Software ohne Bereitstellung weder Freie Software noch Open Source Software ist.

subjektwissenschaftliche Lerntheorie nach Holzkamp und weitere wichtige Aspekte des Lernens beschrieben, die für die spätere Diskussion der Ergebnisse relevant sind. Im dritten Kapitel wird das methodische Vorgehen detailliert beschrieben. Anschließend werden die Ergebnisse anhand einer konkreten FLOSS-Entwicklerkarriere dargestellt, die in der Analyse einer idealtypischen Entwicklungsgeschichte mündet. Auf die bisherige Analyse aufbauend wird dann eine Grounded Theory des Phänomens diskutiert. Das letzte Kapitel fasst die Ergebnisse zusammen und soll darüber hinaus noch einen kurzen Ausblick auf mögliche weitere Forschungsrichtungen gegeben.

2. FLOSS - Geschichte, Strukturen und Philosophie

2.1 Die Geschichte der Free/ Libre Open Source Software

Auf die Frage was „Free/ Libre Open Source Software" (FLOSS) eigentlich ist und wofür es steht, lohnt es sich, den Begriff erst einmal Wort für Wort im geschichtlichen Verlauf zu durchleuchten. Bis Ende der 50er Jahre galten Soft- und Hardware als eine Einheit und wurden auch so vertrieben. Unter Hardware versteht man alle physischen Bestandteile des Computers angefangen bei den Kabeln über den Prozessor bis hin zu den Datenträgern. Bei der Vervielfältigung von Hardware fallen hohe Kosten für Rohstoffe und Entwicklung an, wohingegen für das Kopieren von fertiger Software, wenn man es heutzutage nicht direkt digital verschickt, gerade einmal der Preis eines Datenträgers anfällt und dessen Transport zum Kunden. Bis Ende der 1960er Jahre war Software eine kostenlose Beigabe zur Hardware und wurde mit einsehbarem und modifizierbarem Quellcode (engl. Source Code) geliefert, wodurch sie zur Spielwiese für Wissenschaftler[2] und „Hacker[3]" avancierte, die Fehler und Verbesserungen nicht nur untereinander austauschten, sondern auch wieder an die „Autoren" zurückgeben. Dadurch wurden die Programme benutzerfreundlicher, fehlerfreier und entwickelten sich schneller. Im Juni 1969 entschied IBM als erstes Unternehmen, ihre Dienstleistungen zur Wartung und Weiterentwicklung von Software getrennt von der Hardware anzubieten und eröffnete dadurch einen neuen Wirtschaftszweig. In den 1970ern wurde Software dann zum Geschäftsgeheimnis und damit proprietär und unfrei. Um ihr Monopol zu sichern, änderten die Softwareunternehmen bis Ende der 1970er die Nutzungsbedingungen und erstellten restriktive Lizenzen: Neben der eingeschränkten Verfügbarkeit und Einschränkungen zum Kopieren gaben sie den Quellcode nicht mehr heraus und verhinderten dadurch auch die Modifikation der Software. Zusätzlich überzeugten sie einige der Hacker, für sich zu arbeiten, was von einigen als Verrat empfunden wurde. Es entwickelten sich verschiedene Blickwinkel, wie nicht-proprietäre Software zu entwickeln sei.

[2] Wenn die männliche Form verwendet wird, ist immer gleichermaßen auch die weibliche gemeint.
[3] Der Begriff Hacker war ursprünglich die selbst-referenzielle Bezeichnung der Pioniere der Softwareentwicklung vom MIT und sollte nicht verwechselt werden mit *Crackern*, welche böswillig Sicherheitssysteme knacken

2.2 Commons-based Peer Production[4]

Von Beginn an entwickelte sich bei der Softwareentwicklung ein zugleich sozio-ökonomisches als auch sozio-technisches Produktionssystem[5], eine *Common-based Peer Production* (Benkler & Nissenbaum 2006). In einem solchen System[6] verbinden sich Einzelne zu Arbeitsgruppen von 10 bis mehreren 100.000 Leuten und stellen durch ihre Mitarbeit ihre Informationen, ihr Wissen und ihr Kulturgut der Gemeinschaft als gemeinsam genutzte Ressourcen zur Verfügung. Den Mitwirkenden geht es weder um finanziellen Ausgleich noch um starre, zentralisierte Hierarchien und dennoch könnte man sie durchaus als effektiv bezeichnen. Diese Wissensallmende[7] ermöglicht, dass rechtschaffendes Verhalten in der Praxis nicht nur vorgeführt sondern auch erlebt werden kann (Benkler & Nissenbaum 2006: 395).
Die Kerncharakteristika der Common-based Peer Production, sind die Dezentralisierung und dass sie sich selbst durch soziale Hinweise (cues) und soziale Motivation organisieren und motivieren. Es gibt drei strukturierende Merkmale: Als erstes ist die baukastenartige (modulare) Zusammensetzung der einzelnen Projekte als auch die der Mitwirkenden zu nennen. Ohne Druck trägt jeder so viel zum großen Ganzen bei, wie Fähigkeiten, Interesse und Möglichkeiten es zulassen. Das zweite Merkmal ist die Größe der Projektmodule, die entscheidend für die Vielfalt und Effektivität des Programms, sowie auch die Erwartungen an und die Motivation für ein Projekt sind. Das dritte strukturierende Merkmal ist ein effizienter Mechanismus, der zum einen die kostengünstige Integration aller Komponenten der Projekte ermöglicht, zum anderen die Qualität der Beiträge kontrolliert und darüber hinaus das Projekt gleichzeitig vor Inkompetenz und böswilligen Beiträgen schützt. Um dieses soziale Produktionssystem der Allmendefertigung durch Gleichberechtigte zu erhalten und zu fördern, wurden in den Projekten verbindliche Regeln, sogenannte Lizenzen, aufgestellt (Benkler & Nissenbaum 2006: 400f).

[4] deutsch: Allmendefertigung durch Gleichberechtigte
[5] wovon sich später die proprietäre Software dann entfernt hat (siehe Kapitel 2.1).
[6] was aus Mangel an Computerzeit an den Universitäten entstand und beflügelt wurde durch das Internet
[7] Definition Wissensallmende: gemeinsames Gut (Allmende) der modernen Informationsgesellschaft

2.3 Die Philosophien hinter FLOSS

2.3.1 Free Software Foundation

Richard Stallman, seit den 1970ern aktiv und einer der ersten Hacker vom Massachusetts Institute of Technology (MIT), sah die Entwicklung von proprietärer Software als freiheitsgefährdend für Entwickler, Nutzer und die Gesellschaft. Als auch das MIT auf ein proprietäres Betriebssystem (unfreies Unix) umsteigen wollte, eröffnete Stallman 1983 als Alternative zu Unix das GNU-Projekt und gründete 1985 die Free Software Foundation (FSF)[8]. Das Wort *free* bezieht sich hier jedoch nicht auf das englische *free* (dt.: gratis) wie in *Freeware*[9], sondern umfasst zusätzlich noch den Aspekt der Freiheit. Die GNU (Gnu´s not Unix) Public Licence, oft nur GPL genannt, beinhaltet das Copyleft-Prinzip nach dem eine einmal einem Projekt zugeteilte Lizenz nicht mehr vom Projekt zu lösen ist. Sie enthält im Grunde die Freiheiten, welche den ersten Hackern im Laufe der 1970er und 1980er Jahre genommen wurden. Stallman hat dazu vier Freiheiten formuliert: Es soll möglich sein die Software zu nutzen, ihre Funktionsweise zu studieren, die Software für die Anpassung an eigene Bedürfnisse zu verändern und die Software in ursprünglicher oder veränderter Form zu verbreiten. Der erste und dritte Freiheitsbegriff sind elementar. Wird der Zugang zum Quellcode und damit das Modifizieren nicht gewährt, gilt die Software als proprietär und unfrei. Wissen wird als unerschöpfliche Ressource verstanden, welches der Gemeinschaft gehört. Diese Grundsätze stützen sich auf moralische und philosophische Werte und zeigen ein Streben nach einer besseren, gerechteren Welt für die Anwender. Zusammenarbeit und Kontrolle über die Software werden als ein wichtiges ethisches, soziales und politisches Bedürfnis gesehen. Das Programm ist nur ein Werkzeug, um dieses Ziel zu erreichen. Die Freiheiten sollen dieses Ziel einer besseren und gerechteren Welt bewahren.[10]

[8] http://www.gnu.org/gnu/gnu-history
[9] welche ohne einsehbaren Source Code ist, aber je nach Lizenz kopiert und weiter verbreitet werden darf.
[10] http://www.gnu.org/philosophy/free-sw

2.3.2 Open Source Initiative

Anfang 1998 distanzierten sich einige Entwickler vom Begriff „Freie Software" (FS) und gründeten die Open Source Initiative (OSI). Das freie, Unix-ähnliche Betriebssystem GNU/Linux wurde seit 1992 bereits zur kommerziellen Veröffentlichung freigeben. Um im Wettbewerb mit proprietärer Software bestehen zu können und mögliche Investoren anzulocken, die der Begriff „Freie Software" und dessen Ideologie bisher abschreckte, wurde der Marketing-Begriff „Open Source Software" (OSS) eingeführt.[11] Die Open Source Definition[12] ist im Grunde deckungsgleich mit jener der Freien Software, wobei Open Source die Praktikabilität des offenen Quellcodes als Begründung höher schätzt als freiheitliche Ideale, ganz nach dem Motto: Wenn alle daran mitarbeiten können, kommt am Ende ein besseres Produkt dabei heraus. Im Prinzip kam es zu der Trennung von FS und OSS durch einen Streit um die Priorität von Effektivität[13] oder Effizienz[14]. Dieser Streit zwischen den beiden Idealen spielt sich mehr auf philosophischer bzw. Marketing-technischer als auf alltäglicher Ebene ab. In der konkreten Arbeit gibt es viele Kooperationen zwischen den beiden Lagern, was an den Lizenzen deutlich wird. Im Mittelpunkt der OSS steht eine aus der Perspektive der Entwickler effiziente Arbeitsweise. Trotz der Reibungspunkte sehen sich FSF und die OSI nicht als Gegner, denn das ist aus Stallmans Sicht die proprietäre Software[15]. Beide Bewegungen versuchen die Strukturen der Common-based Peer Production zu erhalten. Um beide Arten von Software unabhängig und unvoreingenommen zu benennen, empfehlen beide Lager, gerade für wissenschaftliche Texte den Begriff Free Open Source Software (FOSS) oder Free/Libre Open Source Software (FLOSS) zu verwenden. Das L, wie livre oder libre soll Stallmans Erläuterung von Freiheiten widerspiegeln und wiederum vom Begriff kostenlos wegführen. In diesem Forschungsbericht werden wir daher die produktions-orientierte OSS und die ideologie-geladene Freie Software, die in der Praxis oft fälschlicherweise wie Synonyme verwendet werden, unter dem Begriff FLOSS zusammenfassen.

[11] http://www.gnu.org/gnu/gnu-history
[12] http://opensource.org/docs/osd
[13] Wirksamkeit - doing the right thing
[14] Wirtschaftlichkeit - doing things right
[15] http://www.gnu.org/philosophy/open-source-misses-the-point

3. Definitionen theoretischer Grundbegriffe

Dieses Kapitel soll einen kurzen Überblick über die für die Interpretation der Daten verwendeten theoretischen Konzepte liefern. Hierfür werden die für unsere Analyse relevanten Begriffe eingeführt und außerdem auf weiterführende Literatur hingewiesen, ohne den Anspruch, die angesprochenen Theorien vollständig darzustellen.

3.1 Lernen aus subjektwissenschaftlicher Perspektive

Holzkamp legte die Grundlegungen der subjektwissenschaftlichen Lerntheorie fest und konzipierte damit eine moderne Ansicht des Lernens aus der Sicht des Subjekts. Das Subjekt sieht die Welt aus seiner Perspektive und nimmt sie intentional wahr (Holzkamp 1995). Das Subjekt bildet somit das *Intentionalitätszentrum* (Holzkamp 1995: 21) seiner Lebenswelt und nimmt andere Subjekte als ihre eigenen *Intentionalitätszentren* wahr. Die Intentionalität impliziert Absichten, Pläne und Vorsätze, die das Subjekt vom Standpunkt seiner Lebensinteressen aus bestimmt. Das Subjekt ist also kein neutraler Beobachter, sondern ein „sinnlich-körperliches, bedürftiges, interessiertes Subjekt" (Holzkamp 1995: 21). Die aus seinem Standpunkt aus wahrgenommene Welt ist für das Subjekt bedeutend und jedes Subjekt handelt aus seiner Perspektive vernünftig.

Aus der subjektwissenschaftlichen Perspektive sind Handlungen nicht determiniert, sondern das Subjekt besitzt ein Repertoire an *Handlungsmöglichkeiten*, die es bewusst beeinflussen kann. Die *Handlungsmöglichkeiten* bzw. *Handlungsgründe* sind jedoch schicht- und klassenspezifisch. Sie werden durch die aktuelle Lebenssituation konstruiert. Die *Handlungsgründe* – das Verhalten – spiegeln somit den Standpunkt des Subjekts wider. Das Subjekt kann bedingt durch seine *Lebensinteressen* entscheiden, welche *Handlungsmöglichkeiten* es ergreift oder verweigert. Dabei muss jedoch beachtet werden, dass auch kausale Gründe zum Handeln führen können – sie sind also nicht immer vernünftig oder begründet beziehungsweise unvernünftig oder unbegründet. Dementsprechend sind Handlungen Realisierungen von Bedeutungen und die *Intentionalität* nicht nur ein kognitiver und mentaler Vorgang, sondern ein aktives Umgestalten von Lebensbedingungen. Die Gründe, warum ein Subjekt handelt bzw. konkret lernt, können

verschiedene Auslöser haben, die Holzkamp als *Lernmotivationen*[16] bezeichnet. Holzkamp unterscheidet zwischen zwei Lernmotivationen: dem *defensivem* bzw. *expansivem Lernen,* welche beide für diese Studie relevant sind.

3.1.1 *Defensives* und *expansives Lernen*

Der Unterschied zwischen dem *defensiven* und dem *expansiven Lernen* liegt darin, dass das *defensive Lernen* von außen – also nicht vom Subjekt ausgehend – ausgelöst wird, das *expansive Lernen* jedoch vom Subjekt ausgeht (Holzkamp 1995). Daraus folgert er, dass das *defensive Lernen* mit möglichst geringem Aufwand erledigt wird. Im Kontrast zum *defensiven Lernen*, entsteht das *expansive Lernen* durch eine *Diskrepanzerfahrung,* die das Subjekt in seinem Leben macht. Durch die *Diskrepanzerfahrung* wird das Lerninteresse des Subjekts geweckt und das Subjekt fühlt den Wunsch, seine *Handlungsmöglichkeiten* zu erweitern. Das bewusste Umgestalten der Lebensbedingungen erfolgt somit durch die Lernmotivation des expansiven Lernens.

Das *expansive* bzw. das *defensive Lernen* zeigen Gemeinsamkeiten mit der *intrinsischen* bzw. *extrinsischen Motivation*. Weiner verweist in seinem Buch „Motivationspsychologie" auf die Definition extrinsischer bzw. intrinsischer Motivation von Rotter und de Charms, die die intrinsische Motivation als „internale Lokation der Kontrolle" und die extrinsische Motivation als „externale Lokation der Kontrolle" beschreiben (Weiner 1994: 203). Bei der intrinsischen Motivation wird also „eine bestimmte Tätigkeit um ihrer selbst willen" (Schürmann 2013: 31) ausgeführt, da das Interesse alleine ausreicht, damit die Handlung fortgesetzt wird. Im Gegensatz dazu ist bei der extrinsischen Motivation eine Beeinflussung, wie Belohnungen oder Bestrafungen, nötig, damit das Individuum handelt.[17] Barbuto (1998) konzipierte eine weitere

[16] Es gibt verschiedene Ansichten zu Motivation: In der humanistischen Motivationstheorie wird sie „als Begriff der Gesamtheit aller Motive bezeichnet, die der Verwirklichung von Lebens-, Bedeutungs- und Sinnwerten dienen" (Schürmann 2013: 30). Andere Bereiche, wie einige psychologische Theorien sehen die Motivation als angeborenes Verhalten und als ein automatischer eingeleiteter Aktivierungsprozess, der von äußeren Reizen und inneren Bedürfnissen abhängig ist (Schürmann 2013). Eine weitere Sichtweise ist das Motivationen eine kognitive Repräsentation von gewünschten Zuständen sind.

[17] Weiner beschreibt einen weiteren interessanten Aspekt. Die intrinsische Motivation kann verloren gehen, sobald eine extrinsische einfließt. Dies wurde an einem Versuch mit Kindern bewiesen (Weiner 1994).

Ebene, welche die intrinsische und extrinsische Motivation differenzierter beschreibt und die nun im Folgenden dargestellt werden soll.

3.1.*2 Intrinsic process motivation* und *internal self-concept-based motivation*

Barbuto unterteilt die intrinsische Motivation in *intrinsic process motivation* und *internal self-concept-based motivation.* Die *extrinsische Motivation* teilt Barbuto in *instrumental Motivation, external self-concept-based motivation* und *goal internalization motivation* auf (Barbuto et al. 1998). Relevant für diese Studie sind besonders die *intrinsic process motivation* sowie die *internal self-concept-based motivation,* da diese weiterführend die intrinsischen Motivationen beschreiben. Die *intrinsische Motivation* wiederum kann – wie schon erwähnt wurde – einerseits mit dem *expansiven Lernen* in Verbindung gebracht werden, andererseits nimmt sie eine explizite Rolle in der Grounded Theory ein. Infolgedessen werden diese beiden *Lernmotivationen* ausführlicher beschrieben.

Die *intrinsic process motivation* (dt: *intrinsische Prozessmotivation*) findet dann statt, wenn der Anreiz der Tätigkeit selbst dem Individuum Freude bereitet, dabei steht der Spaß als Hauptmotivation im Vordergrund. Die *internal self-concept-based motivation* (dt. *Internes Selbstverständnis*) motiviert das Individuum durch seine Ideale. Darunter zählt Barbuto Eigenschaften, Kompetenzen und Werte, mit denen sich das Individuum identifiziert: „In this source, motivation is internally based when the individual is inner-directed. The individual sets internal standards of traits, competencies and values that become basis of the ideal self" (Barbuto et al. 1998: 1013). Das Individuum wird dadurch zum Handeln motiviert, dass es eine höhere Stufe seiner Fähigkeiten („standards" s. o.) erreichen möchte.

3.2 *Learning by doing* und *explorative learning*

Die Studie „Toward an Understanding of the Motivation of Open Source Software Developers" von Kishida und Ye (2003) untersuchte Motivationen von OSS-Entwicklern. Sie sehen die intrinsische als eine kognitive und die extrinsische als eine soziale Motivation. Dabei kamen sie zu dem Ergebnis, dass eine Community, die die OSS-Entwickler bei der Kommunikation untereinander unterstützt, einen wichtigen Punkt darstellt (extrinsische Motivation).
Für diese Studie jedoch bedeutender ist die Tatsache, dass das Lernen als intrinsische Motivation eine zentrale Rolle spielt. Signifikant ist, dass die Lernenden eine innere Zufriedenheit fühlen, wenn sie beispielsweise an einem OSS Projekt mitarbeiten. Das Lernen bietet diese intrinsische Zufriedenheit für OSS-Entwickler. Es kristallisierten sich zwei Lernarten heraus: das *learning by doing* und das *explorative learning*. Das *explorative learning* ist ein erforschendes Lernen, so wie es auch bei Wissenschaftlern zu finden ist. Die Motivation, ein schon vorhandenes Problem zu lösen oder einen neuen Weg zu finden, um Dinge tun zu können, kann intrinsische Gründe haben, aber auch aus Notwendigkeit durch äußere Umstände (extrinsische Motivation) entstehen. Das explorative Lernen ist hingegen meist bei Entwicklern zu finden, die sich schon viel Wissen angeeignet haben und Projekte starten, um etwas Neues zu lernen.
Das *learning by doing* setzt eine Freude am Erforschen der existierenden Programme und der Möglichkeiten, sie zu verbessern, voraus. Der OSS-Entwickler setzt sein aktuelles Wissen ein und erweitert es gleichzeitig.

4. Forschungsvorgehen: Methodik und Empirie

Für diese Studie wurden narrative Interviews (Przyborski & Wohlrab-Sahr 2009: 92ff) als Erhebungsmethode ausgewählt. Der Eingangsstimulus „Erinnern Sie sich zurück, wie Sie das erste Mal davon[18] gehört haben, wie hat sich Ihr Engagement dafür entwickelt und was hat Sie dazu bewegt, weiter zu machen?" sollte die Interviewten dazu bringen, über ihre freiwillige Tätigkeiten im Bereich FLOSS zu erzählen. Der Eingangsstimulus wurde authentisch formuliert, damit ein natürliches und nicht aufgesetztes Gespräch [19] zustande kommt, bzw. der Interviewpartner ein angenehmes Gefühl hat und keine Hemmungen zum Erzählen entwickelt. Deswegen wurde der Eingangsstimulus je nach Interview-Situation angepasst bzw. erweitert. Die Interviewpartner wurden im Vorgespräch darauf hingewiesen, dass bei diesem Interview die unentgeltliche Tätigkeit als Free, bzw. Open Source Software Entwickler im Fokus steht. Die Interviewpartner wurden nach dem Prinzip des Theoretical Sampling ausgesucht, einer Methode, die im Rahmen der Grounded Theory Methode entwickelt wurde (Przyborski & Wohlrab-Sahr 2009). Da im Rahmen des Lehrforschungsmoduls eine komplette theoretische Sättigung nicht zu erwarten war, wurde von Anfang an Wert darauf gelegt, möglichst unterschiedliche Interviewpartner zu gewinnen. Die Auswahl erfolgte also um eine maximale Kontrastierung[20] und somit eine möglichst hohe Aussagekraft der Daten zu gewährleisten. Als erster Interviewpartner wurde ein „Einzelkämpfer" ausgewählt, der vollkommen unabhängig von anderen Projekten eine eigene Homepage betreibt, auf der er selbst entwickelte Lernprogramme zur Verfügung stellt. Der zweite Interviewpartner wurde über eine in der FLOSS-Szene stark genutzten Plattform aufgrund seines dortigen Engagements und der dort ersichtlichen Zusammenarbeit mit anderen gesucht und angeschrieben. Der dritte Interviewpartner sollte sich noch im Studium befinden. Jedes durchgeführte Interview wurde zunächst transkribiert und anschließend bereits mit dem offenen Kodieren begonnen, um die so entstandenen Kenntnisse

[18] Hier wurde in den Vorgesprächen nicht durchgängig klar zwischen Open Source und freier Software unterschieden, da diese Unterscheidung nicht im anfänglichen Forschungsinteresse lag. Aufgrund eines starken Rückbezugs einiger Interviewten auf die Philosophie, bzw. Ideologie hinter den verschiedenen Formen wurde die Unterscheidung dann erst nachträglich in die Analyse mit eingebracht.
[19] Nicht zu verwechseln mit dem „Authentischem Gespräch" (Przyborski & Wohlrab-Sahr 2009:155).
[20] Die maximale Kontrastierung soll ein Feld eingrenzen bzw. es erschließen, bis keine neuen Erkenntnisse mehr erscheinen. Im Gegensatz dazu überprüft die minimale Kontrastierung eine schon vorhandene Hypothese (Przyborski & Wohlrab-Sahr 2009:177).

des Forschungsfeldes in die weitere Auswahl der Interviewpartner einfließen zu lassen. Nach dem Kodieren der Transkripte zeigte sich, dass der ideologische Hintergrund ebenfalls eine interessante und nicht unwichtige Rolle in der Entwicklung von öffentlichem Quellcode spielt. Dementsprechend kam bei dem letzten Interviewpartner das Kriterium der Software-Philosophie hinzu. Außerdem kristallisierte sich nach dem zweiten Interview noch eine weitere spezifische Richtung heraus, in welcher der vierte Interviewpartner gesucht wurde. Konkret ging es hierbei um die Abgrenzung zu entgeltlicher Arbeit. Deswegen wurde als vierter Interviewpartner jemand gewählt, der die Gedanken der FLOSS-Philosophie kennt und selbst FLOSS-Kurse anbietet. Somit verdient er zwar nicht direkt mit seinem Engagement für Freie Software Geld, aber durchaus mit dem Wissen, das er sich durch die Beschäftigung mit FLOSS angeeignet hatte.

Das Alter[21] der vier männlichen Interviewpartner liegt zwischen 25 und 50 Jahren. Das anfängliche Hauptkriterium bei der Auswahl der Interviewpartner war deren Engagement im Bereich der FLOSS-Entwicklung. Die spezifischen Auswahlkriterien wurden bereits im Abschnitt über das Vorgehen nach dem Theoretical Sampling erläutert. Hier soll nun noch ein kurzer Überblick über die Interviewpartner gegeben werden, um ein paar Informationen über den Hintergrund der Personen zu geben, die nötig sind, um ein Gefühl für die Bedingungsmatrix[22] (Strauss & Corbin 1996:132ff) der interviewten Personen gewinnen zu können.
Der erste Interviewpartner Gerhard P.[23] (Transkriptkürzel Am)[24] ist Ende 30 und Lehrer. Er unterrichtet in den Fächern Biologie und Informatik in der Mittelstufe. Er hat einige Lernprogramme für Schüler und andere Lehrer erstellt, die online zugänglich sind. Der zweite Interviewpartner Giovanni S. (Transkriptkürzel Bm) ist Ende 20 und Physikstudent, der sozial engagiert ist und eigentlich Psychologie studieren wollte. Der dritte Interviewte Eugen V. (Transkriptkürzel Cm) ist Mitte 20, studiert Informationstechnik im Master of Science und arbeitet als Systemadministrator. Der letzte Interviewpartner Dieter V. (Transkriptkürzel Dm) ist

[21] Das Alter und sowie das Geschlecht haben kein Auswahlkriterium dargestellt.
[22] Die Bedingungsmatrix ist eine Methode das komplexe Umfeld einer Person, das die Handlungsoptionen einer Person beeinflusst, in verschiedene Ebenen zu gliedern. Dies wird in dieser Studie zwar nicht in aller Ausführlichkeit durchgeführt, jedoch weist die entwickelte Grounded Theory durchaus eine ähnlich gelagerte Schichtung in verschiedene Bereiche auf.
[23] Der Name wurde ebenso wie alle weiteren Namen geändert.
[24] Das Transkriptkürzel steht jeweils am Ende eines jeden Transkript-Zitats und wird mit der zugehörigen Zeilenangabe versehen. Hieraus lässt sich ersehen aus welcher Phase des Interviews die zitierte Passage stammt.

schätzungsweise Mitte 40 und ist ein studierter Ingenieur, der mittlerweile eine Vielzahl von FLOSS-Kursen anbietet. Es ist auffällig, dass alle Interviewten aus dem Bereich der Natur- und Ingenieurwissenschaften kommen. Dies lässt vermuten, dass bei diesen Personen ein generelles Grundinteresse am Verständnis von technischen Zusammenhängen gegeben ist, welches förderlich für eine Beschäftigung mit FLOSS ist.

In der Analyse der Daten wurden zwei verschiedene, sich gegenseitig ergänzende Wege beschritten. Zunächst wurde versucht, durch die üblichen Analyseschritte[25] der Grounded Theory-Methode eine Grounded Theory zu erstellen. Nach anfänglichen Schwierigkeiten, den zeitlichen Aspekt der Etablierung der FLOSS-Entwicklung in die Grounded Theory miteinzubeziehen, wurde eine FLOSS-"Karriere" pro Interview angefertigt, die anschließend in ein Idealtyp nach Weber überführt wurde (Przyborski & Wohlrab-Sahr 2009: 328 ff). Dadurch war genügend konzeptuelle Vorarbeit geleistet, um auch in den bis dahin entwickelten Kategorien den Veränderungscharakter wiederzuerkennen und eine passende Grounded Theory zu erstellen.

[25] Gemeint sind hier offenes (Strauss & Corbin 1996:75ff) und axiales Kodieren (Strauss & Corbin 1996:94ff), sowie gegen Ende der Analyse, sobald die Kategorien genügend ausgearbeitet waren, auch das selektive Kodieren (Strauss & Corbin 1996:94ff).

5. Analyse des prozeduralen Aspekts der Entwicklung von FLOSS

5.1 Analyse einer FLOSS-Entwicklerkarriere

> *„Em (.) ich war Schüler und brauchte ein neues Betriebssystem und konnte mir kein Windows leisten (.) dann hab ich geguckt (.) was ist kostenlos (.) das war Linux, das war damals Suse glaube ich noch oder war das Redhat (.) irgendeine Linux-Distribution (.) dann hab ich das installieren wollen (.) dann liefen einige Kleinigkeiten nicht (.) irgendwann (.) klar (.) dann liefen einige Kleinigkeiten nicht (.) dann hab dafür (.) dann hab ich halt angefangen zu suchen (.) wie ich das lösen konnte (.) und dann fand ich irgendwann heraus ohh (.) der Quellcode ist ja offen (.) @(.)@ //@(.)@// und dann hab ich da angefangen zu wühlen" (Cm:225-232)*

Dieser kurze Textabschnitt aus dem Interview von Eugen V. zeigt bereits einige wichtige Elemente wie zunächst unbedarfte Laien dazu kommen können, sich mit Open Source zu beschäftigen. Als erstes lässt sich eine konkrete Notwendigkeit feststellen, in diesem Fall die Notwendigkeit, einen Computer mit einem funktionierenden Betriebssystem auszustatten. Die zweite aufscheinende Notwendigkeit besteht durch eine möglichst kostengünstige oder gar kostenlose Lösung zu finden. Diese beiden Notwendigkeiten führen zu der Handlung, sich eine kostenlose Linux-Distribution zu besorgen und zu installieren. Aus der Tatsache, dass wie berichtet einige Kleinigkeiten nicht liefen, ergab sich die dritte Notwendigkeit, die nicht funktionierenden Teile zu verbessern.

Es lässt sich also zumindest für diesen Einzelfall bereits klar darstellen, dass jeweils konkrete Notwendigkeiten als Handlungsauslöser fungierten. An dieser Stelle ist jedoch noch der Hinweis nötig, dass keine dieser Notwendigkeiten in einem determinierenden Verhältnis zur anschließenden Handlung steht. Es lassen sich für jede dieser Notwendigkeiten auch alternative Handlungen als Vergleichshorizonte vorstellen. So wäre es beispielsweise möglich gewesen, sich Windows zum Geburtstag zu wünschen, dafür zu arbeiten oder sich eine Raubkopie zu besorgen. Es scheint also noch weitere Faktoren zu geben, die eine Beschäftigung mit Open Source Software unterstützen. Doch bevor nun diese Frage in den Fokus rückt, soll noch ein Blick auf bestimmte Charakteristika des Programms selbst geworfen werden. Würden nicht bereits

verschiedene kostenlose Linux-Distributionen existieren, könnten sie auch nicht den Bedarf nach einen kostenlosen Betriebssystem decken. Diese Feststellung mag auf den ersten Blick trivial erscheinen jedoch wird diese Sichtweise - das Geschehen vom Programm und seiner Funktionalität her zu betrachten - notwendig, um die weiteren Schritte noch etwas besser verstehen zu können. Der letzte Teil des Zitats deutet dies schon ganz deutlich an: *„und dann fand ich irgendwann heraus ohh (.) der Quellcode ist ja offen (.) @(.)@ //@(.)@// und dann hab ich da angefangen zu wühlen"*. Hier wird zunächst das Erstaunen zum Ausdruck gebracht, dass es sich um offenen Quellcode handelt, der also einsehbar und modifizierbar ist. Die Software schafft somit fast unendliche Handlungsoptionen. Jeder Teil kann in beliebiger Art und Weise ohne prinzipielle Zugangsbeschränkung modifiziert werden. Konkrete Zugangshürden ergeben sich jedoch aus der Komplexität des Quellcodes, der wiederum auf einer Programmiersprache aufbaut. In dem Ausdruck „*angefangen zu wühlen"* zeigt sich der Umgang mit der anfänglichen Unübersichtlichkeit des Quellcodes, die dem Laien erscheinen mag wie die *Suche nach der Nadel im Heuhaufen*, die aufgrund von mangelnder Information zum Aufenthaltsort der Nadel zunächst einem unkoordinierten „*wühlen"* gleichkommt. Wenig später im Interview zeichnet Eugen V. ein ähnliches Bild:

> *„Erst mal konnte ich überhaupt nichts machen (.) erst mal wusste ich überhaupt nicht was dass alles war (.) das waren die erste Schritte in die Informatik (.) der erste Schritt in also (.) in die Programmierung (.) und zuerst habe ich natürlich alles kaputt gemacht @(.)@ @weil ich konnte ja alles verändern (.) aber im Laufe der Zeit (.) @wurden die Veränderungen sinnvoller@" (Cm:235-239)*

Hier zeigt sich wieder das gleiche Element, dass er nämlich als Laie bei seinen ersten Versuchen zwar überall etwas verändern konnte, aber da ihm spezifische Programmierkenntnisse fehlten zunächst nur dazu in der Lage war *alles kaputt zu machen.* Spannend ist nun die Frage, wie es anschließend dazu kam, dass die ersten Veränderungen *„sinnvoll"* und somit wirklich eine Antwort auf die ursprüngliche Notwendigkeit wurden. An dieser Stelle steht nicht viel dazu als Erklärung, sondern nur ein *„aber im Laufe der Zeit"*. Dieser Ausdruck weist jedoch bereits auf einige weitere wichtige Aspekte hin. Es handelt sich um eine Veränderung, die einige Zeit in Anspruch genommen hat. Diese Veränderung findet primär natürlich auf der Ebene des

Quellcodes statt, also von zunächst *sinnlosen* Quellcode-Änderungen zu *sinnvollen* Quellcode-Änderungen. Doch die Veränderungen auf der Ebene des Quellcodes wären nicht möglich ohne eine grundlegende Veränderung auf Seiten des handelnden Subjekts. Ohne einen signifikanten Zuwachs an Programmierkenntnissen und an Einblick in die Struktur des Programms wären keine willentlichen, sinnvollen Änderungen möglich. Ein weiterer Grund warum diese Ebene nicht explizit angesprochen wird, könnte sein, dass sich Eugen V. gar nicht voll bewusst ist, was für einen großen Wissenszuwachs er sich in dieser Phase erarbeitet hat. Es scheint sich bei Eugen V. eher um eine Form des *learning by doing* zu handeln, als um ein institutionell geprägtes lernen über Bücher oder Kurse. Hierauf wird im nächsten Kapitel im Abschnitt zur Wissensaneignung noch ausführlich eingegangen.

Nun fehlt jedoch noch der entscheidende Schritt für die Entwicklung von Open Source Software, der in der Veröffentlichung der gemachten Veränderungen besteht. Alle bisher angesprochenen Veränderungen fanden nur lokal auf dem Rechner statt. Eugen V. beschreibt diesen Vorgang wie folgt:

> *„Also (.) das war eigentlich immer so: viel wie ich Zeit und Lust hatte (.) em also wenn ich irgendwo irgendwas mitgearbeitet habe (.) irgendwas mit entwickelt habe (.) waren das meist irgendwelche Kleinigkeiten (.) weil ich jetzt in den letzten Jahre ( ) mit meinem Studium beschäftigt war (.) und wenn z=zum Beispiel habe ich eine Kleinigkeiten unter Ubuntu geschrieben (.) Treiber bisschen modifiziert und so Kleinigkeiten (.) veröffentlicht dann wurde das irgendwann wahrscheinlich geprüft oder für gut oder f=für f=falsch befunden hat mich auch nicht weiter interessiert (.) weil es es lief letzten Endes bei mir und (.) für mich ähm (2) wars b-( ) Open Source einfach der Umstand (.) ich konnte man kann (.) seine Gedanken mit einbringen (.) ohne dass man sich d=da weiter für interessieren braucht (.) wenn man ne kurze Idee hat (.) oder irgendwas irgendne Verbesserung hat (.) kann °Ver-°besserung ähm umsetzen (.) dann (.) bindet man die Verbesserung mit ins Projekt ein und braucht sich n=nich mehr weiter drum scheren (.) das war für mich immer das Wichtige an Open Source" (Cm:69-82)*

Eugen V. benennt die gemachten Veränderungen als Kleinigkeiten im Programm, also kleine Adaptionen eines bestehenden großen Projekts, dass von anderen Entwicklern geschaffen und zur Verfügung gestellt wurde. Er stellt mehrfach klar, dass es ihm wichtig war, dass sich keine weiteren Verpflichtungen aus seinem Handeln ergeben. Er sagt sogar explizit, dass es ihn auch nicht „weiter interessiert" hat, ob seine Veränderungen in das bestehende Projekt aufgenommen wurden, da sein primäres Ziel, eine Veränderung seiner lokal genutzten Software, ja bereits erreicht wurde. Auf den ersten Blick könnte man sich fragen, warum er überhaupt den Schritt macht seine Änderungen den Verantwortlichen für das Projekt zur Verfügung zu stellen. Hinter den Formulierungen *„man kann (.) seine Gedanken mit einbringen"* und *„wenn man ne kurze Idee hat (.) oder irgendwas irgendne Verbesserung hat"* kann man möglicherweise ein partizipatives Element erahnen. Es tritt nicht klar zu Tage, aber durch die Formulierung scheint es nicht so sehr um die konkreten Mit-Entwickler oder End-Nutzer zu gehen, sondern eher auf der Ebene der Software statt zu finden. Dass die eigenen Gedanken und Ideen Eingang finden in die Software, um diese zu verbessern, scheint das primäre Ziel zu sein. Dies zeigt sich auch deutlich an anderer Stelle:

> *„Ich ähm (.) schreibe auch die meisten Code (.) den ich auch veröffentliche (.) unter Pseu=Pseudo:nym (.) denn mir geht es darum (.) wirklich nur darum dieses Tool für mich zu verbessern und auch für alle anderen (2) im Nebeneffekt (.) aber für mich ist dann (.) ja (.) im Moment geht es mir nur darum das ich (.) ir- das Tool irgendwie verbessern kann und ichs auch verbessere (.) weil ich es halt kann" (Cm:263-268)*

Die Tatsache, dass Eugen V. seinen Quellcode meist unter einem Pseudonym veröffentlicht, zeigt auf, dass es ihm tatsächlich wenig um direkte soziale Anerkennung geht, sondern zunächst darum durch das Programm seine direkte Lebensumwelt, zu dem auch sein Rechner gehört, zu verbessern. Der Rest geschieht laut Eugen V. nur *„im Nebeneffekt"*. Im Vergleich zur sehr voraussetzungsreichen Praxis des Entwickelns von Software ist das Online-Stellen der Software nur einen Mausklick entfernt und besitzt deshalb nicht sofort eine ähnliche Bedeutung im Bezugsrahmen des handelnden Subjekts. Nicht einfach gestaltet sich die Analyse der letzten zwei Sätze: *„im Moment geht es mir nur darum das ich (.) ir- das Tool irgendwie verbessern kann"*. Hier lässt sich durchaus ein konkreter Nutzen als zentrales Element herauslesen, jedoch

scheint das Verb „kann" nicht zwangsläufig nötig. Eugen V. könnte genauso gut sagen, dass es ihm darum geht, das Tool irgendwie zu verbessern und damit einfach den konkreten Nutzen der Handlung darzustellen. Stattdessen wählt er zunächst eine Darstellung seiner Handlungsfähigkeit, dass er es kann, wodurch der Nachsatz „*und ichs auch verbessere"* nötig wird, der klarstellt, dass es auch um die konkrete Handlung der Verbesserung eines Tools, also einer kleinen nützlichen Software, geht. Im Anschluss verweist er nochmals auf seine Handlungsfähigkeit „*weil ich es halt kann"*. Dies lässt vermuten, dass er sich seiner eigenen Wirkmächtigkeit durchaus bewusst ist und diese auch einen hohen Stellenwert besitzt. Dieses Moment der eigenen Wirkmächtigkeit zeigt sich viel stärker im ganzen Interview als beispielsweise ein Bewusstsein der Tatsache, dass er sich einen starken Wissenszuwachs an Programmierkenntnissen erarbeitet hat. Es ging ihm nicht darum, etwas zu lernen, um etwas zu wissen, sondern darum, etwas tun zu können. Es ging ihm darum, eine konkrete Handlungsantwort auf die auftretenden Notwendigkeiten geben zu können.

5.2 Idealtyp einer FLOSS-Entwicklungsgeschichte

Nun soll aufbauend auf den in Kapitel 5.1 gewonnenen Erkenntnissen der stark abstrahierte Idealtyp einer *FLOSS-Entwicklungsgeschichte* dargestellt werden, der anschließend noch anhand von konkreten Ausschnitten aus den Interviews in seiner Dimensionalität dargestellt werden soll. Die Entscheidung, eine idealtypische Entwicklungsgeschichte und nicht eine Entwicklerkarriere herauszuarbeiten, fiel aufgrund der vielen Hinweise auf die Produktzentriertheit. In einer personenzentrierten Entwicklerkarriere könnten sich diese produktzentrierten Aspekte nicht adäquat widerspiegeln. Der erste zentrale Punkt in der Analyse des Interviews von Eugen V. war, dass konkrete Notwendigkeiten als Handlungsauslöser fungierten. Der erste Schritt einer idealtypischen Entwicklungsgeschichte soll deswegen nun unter dem Kategorie **Auslöser** dargestellt werden, da noch andere Elemente ebenfalls als Auslöser für eine Handlungskaskade dienen können, an deren Ende dann die Veröffentlichung von FLOSS-Software steht. Der zweite Schritt war ein enormer Wissenszuwachs, ohne den die weiteren Schritte nicht möglich geworden wären. Dieser Schritt soll im weiteren unter dem Begriff **Wissensaneignung** Eingang in die idealtypische Entwicklungsgeschichte finden. Der dritte Schritt betrifft das konkrete Programmieren, worunter sowohl kleine Adaptionen bestehender Programme, als auch die Entwicklung ganz neuer und eigenständiger Software fallen. Auch alle mit dem Programmieren verknüpfte Tätigkeiten, wie z.B. Überlegungen zum Programm-Design, sind hier prinzipiell zu berücksichtigen. Dieser Punkt soll im weiteren **Entwicklung von Software** heißen. Schlussendlich fehlt in der Darstellung nur noch das Moment des Zur Verfügung Stellens der bereits entwickelten Software. Dies geschieht normalerweise über das Internet und kann entweder im Rahmen bereits etablierter Verbreitungssysteme oder über eine eigene Homepage geschehen. Dieser Aspekt soll unter dem Begriff der **Freigabe** subsumiert werden. Schematisch ergibt sich nun folgende Reihenfolge für die einzelnen notwendigen Schritte:

Auslöser ⇒ Wissensaneignung ⇒ Entwicklung von Software ⇒ Freigabe

Diese Art der schematischen Darstellung soll nun keineswegs implizieren, dass es sich um ein lineares Phänomen handelt. Jeder konkrete Verlauf wird eher einem zyklischen Durchlaufen mehrerer Schritte ähneln, die womöglich gar nicht klar zu trennen sein werden. Die Auslöser

können vielfältiger Natur sein und zu verschiedenen Zeitpunkten innerhalb des Verlaufs auftreten und damit die Entwicklung am Laufen halten, so wie dies auch für die dritte Notwendigkeit im Kapitel 5.1 galt. Ohne die Kleinigkeiten, die immer wieder nicht funktionierten, wäre der Lernprozess und Entwicklungsprozess ins Stocken geraten. Ein perfekt funktionierendes System würde nicht dazu einladen, sich mit ihm zu beschäftigen. Jedoch muss ein initialer Auslöser am Anfang gestanden haben und somit erklärt sich die Darstellung des Auslösers als der erste notwendige Schritt. Die nächsten Schritte der Wissensaneignung und die Entwicklung können ebenfalls wie bereits dargestellt in einem *learning by doing* Prozess Hand in Hand gehen und chronologisch nicht klar davon zu trennen sein. Jedoch ist es auch hier notwendig, dass ein Wissenszuwachs vor einer wirklichen Weiterentwicklung der Software stattgefunden haben muss, da die Wahrscheinlichkeit, durch einen reinen „Zufallstreffer" eine funktionierende Weiterentwicklung hervorzubringen, verschwindend gering sind. In den folgenden Unterkapiteln sollen nun die einzelnen abstrakten Schritte in ihrer Dimensionalität betrachtet werden.

5.2.1 Auslöser

Das Element der **Notwendigkeit** wurde bereits bei Eugen V. ausführlich dargestellt und findet sich in ähnlicher Weise in allen Interviews. Dieter V. sagt hierzu beispielsweise:

> *„Dann irgendwann äh Mal für mich zuhause (.) damals in meiner Wohngemeinschaft in der ich damals gewohnt hab (.) haben wir einen Internet-Router gebraucht (.) so dass man gemeinsam halt Internet nutzen kann (.) und das gabs damals so nicht zu kaufen (.) und deswegen haben wir nen alten PC hergenommen und da drauf dann eben halt dann (.) ja der sollten dann immer das Internet-Routing machen und ähm (.) das hab ich erst mal mit Windows probiert und das hat aber nicht so gut geklappt und dann hab ich es eben halt (.) eben halt mit Linux eben halt ahm probiert und das hat dann eben halt funktioniert" (Dm:12-20)*

Die hier dargestellte Notwendigkeit, einen Internet-Router für die Wohngemeinschaft zu installieren, führte im ersten Schritt dazu, es mit Windows zu probieren, was jedoch nicht den gewünschten Erfolg brachte. Hieraus ergab sich dann die Notwendigkeit einer funktionierenden Alternative, die in GNU/Linux gefunden wurde. Doch die Notwendigkeiten, die als Auslöser für die Beschäftigung mit FLOSS dienten, stammen nicht durchgängig aus dem privaten Bereich. Giovanni S. hat sich beispielsweise erstmals mit GNU/Linux beschäftigt, um einem Verein, für den er sich ehrenamtlich engagiert, eine technische Lösung für ein dort konkret auftretendes Problem zu liefern. Gerhard P. seinerseits berichtet von Notwendigkeiten, die sich aus seinem Beruf ergaben:

> *„Und dann eigentlich so auch ungefähr 2004 rum angefangen bei uns die Systemadministration mit zu machen (.) mit nem Kollegen (.) und da //an der Schule jetzt?// an der Schule (.) na gut des sind auch inzwischen ja über hundert Rechner zwei getrennte Netzwerke (.) also des ist eigentlich das was (.) nicht so einmal was installieren sondern schon so richtig (.) ähm und (.) da: drauf gekommen okay diese Linuxgeschichte (.) vorher eigentlich damit nix zu tun gehabt und da dann Shellskripte programmiert weil ma halt dann Sachen automatisieren kann" (Am:27-34)*

Durch die berufliche Veränderung, nun als Systemadministrator mit einem Kollegen zusammen für über hundert Rechner verantwortlich zu sein, ergab sich die Notwendigkeit, sich mit Linux zu beschäftigen. Aus der konkreten Notwendigkeit heraus, manche Abläufe zu automatisieren, was bei einer Anzahl von über hundert Rechnern tatsächlich eine enorme Arbeitserleichterung darstellen kann, hat Gerhard P. begonnen, Skripte zu programmieren. Auch hier erfolgt die Darstellung auf der Ebene der Software, der Punkt der Wissensaneignung, die auf der Ebene der Person stattgefunden haben muss, wird hier nicht thematisiert.

Die Darstellung, dass FLOSS oft als Antwort auf konkrete Notwendigkeiten genutzt wird, ist sicherlich ein zentraler Punkt, der jedoch noch nicht die ganze Bandbreite der Auslöser darstellt. Ein weiterer Auslöser scheint Neugierde zu sein. Dieter V. berichtet hierzu:

„Ja (.) das war dann so der Anfang der Benutzung (.) und ähm zum ersten Mal Code geschrieben (.) dazu musste ich erst mal programmieren lernen (.) ahm (.) das hab ich ahm in (.) der Programmiersprache namens Pearl (.) ahm (2) und das hab ich erst mal nur weil es mich so schon fasziniert hat irgenswie=so Programmieren (.) ich dachte immer es wäre was total kompliziertes (.) und ahm (.) ahm hab aber dann ahm mit=mit dieser Programmiersprache Pearl eben halt die ist eben halt nicht so: kompliziert wie das immer bei anderen Programmiersprachen ist (.) man muss da nicht kompilieren oder irgendwie so komischen Sachen (.) sondern man (.) tippt da irgend ein Befehle ein und dann tut er sofort was" (Dm:34-42)

Dieter V. reflektiert hier etwas ausführlicher als in den bisher genannten Zitaten über die notwendigen Schritte, die dem erfolgreichen Programmieren vorausgehen. So stellt er explizit dar, dass er das Programmieren zunächst lernen musste und gibt als Auslöser dafür an: „*das hab ich erst mal nur weil es mich so schon fasziniert hat irgenswie=so Programmieren*". Die Faszination einer „*total komplizierten*" *Geheimwissenschaft* Programmieren, auf die er neugierig war, scheint eine große Anziehungskraft zu besitzen. Wahrscheinlich haben auch in der in Kapitel 5.1 dargestellten Entwicklerkarriere von Eugen V. Motive wie Neugier eine unterstützende Rolle dafür gespielt, dass nicht alternative Handlungsoptionen wahrgenommen wurden, um den Notwendigkeiten zu begegnen. In diesem Abschnitt konnte gezeigt werden, dass neben reinen Notwendigkeiten auch Neugier und Faszination als Auslöser dafür dienen können, um sich Wissen anzueignen. Wie diese Wissensaneignung genau vonstatten geht, soll der nächste Abschnitt klären.

5.2.2 Wissensaneignung

Wie bereits eingangs kurz erwähnt kann der Lernprozess mit der Entwicklung des Programms verknüpft als sogenanntes *learning by doing* geschehen.

„Ich hatte natürlich auch noch nicht viel Ahnung davon und wusste nicht genau wie es halt funktioniert em (.) aber joa hab das dann halt (.) mich dann so halt (.)

damit gearbeitet (.) und dann eben halt mehr und mehr verstanden wie es halt funktioniert" (Dm:31-34)

In diesem Zitat wird deutlich, dass Dieter V. nach dem Prinzip des *learning by doing* arbeitete. Er hat *„nicht viel Ahnung"*, jedoch scheint er bereits basales Wissen zu besitzen, das er anwendete und dadurch seine Fähigkeiten erweitern konnte: *„mehr und mehr verstanden wie es halt funktioniert"*. Dieses Element findet sich in allen Interviews in unterschiedlicher Ausprägung wieder, aber auch klassische Formen der Wissensaneignung wie beispielsweise das Lesen eines Buchs zum Thema wurden genutzt:

„Und da dann halt gemerkt so och des passt @eigentlich@ ganz gut (.) und dann halt ähm so n Buch entdeckt irgendwie was war das des war son Mathezeug (.) irgendwie Python für Kids (.) und Programmiersprache Python ist relativ klar aufgebaut und da halt gemerkt so oh kann man schöne Sachen mit machen" (Am:46-50)

Doch diese Studie wollte versuchen, nicht nur das konkrete Geschehen nachzuzeichnen, sondern auch die Motivationen hinter dem Handeln untersuchen. Hierzu lässt sich zunächst einmal feststellen, dass die Wissensaneignung ausgelöst wird durch gewisse Lernmotivationen. Die FLOSS-Entwickler zeigen den Willen, das Nichtwissen in Wissen umzugestalten. Dies geschieht durch das Erforschen und das eigenständige Aneignen von Programmierkenntnissen. Giovanni S. erlebte eine starke *Diskrepanzerfahrung,* die ihn dazu bewegte, sich weiteres Wissen anzueignen. Man könnte sogar von einem Leiden an der Situation sprechen:

„Ja ich habe mit wirklich angefangen mit (.) verdammt warum geht das nicht und ich sitz hier vor einem schwarzen kleinen Konsolenfenster und habe keine Ahnung warum das gerade nicht die graphische Oberfläche ist //mhm// @so das erste Frusterlebnis bei der Installation von Linux@ (2) und (3) das ist vielleicht auch das was (.) viele davon abhält sich damit zu beschäftigen weil man viel viel lernen (.) lesen muss" (Bm:502-507)

Doch die *Diskrepanzerfahrung* steht nicht nur am Anfang mit der Beschäftigung mit Linux, sondern tritt immer wieder auf, da sich die einzelnen Bereiche und das dafür benötigte Wissen stark unterscheiden. Hierzu findet sich bei Giovanni S. etwas später im Interview:

> *„Open Source äh (.) ab ins kalte Wasser @(.)@ (4) es gibt halt auch (.) vielleicht das (.) dazu (.) ich hab mich auch extre:m in verschiedensten Bereichen (.) hm meiner Zeit bewecht von (.) Server::sachen die im Open Source Bereich sind hin bis zu Webdienste bis hin zu Programmierung von kleinen (.) äh Zusatzprogrammen die auf dem Desktop laufen also das sind sind ja zum Teil (.) sind die Fähigkeiten sicherlich die man da braucht überschneidend (.) aber zum Teil auch so: vö:llig unterschiedlich so dass ich zum Beispiel für das Projekt was du gefunden hast (.) auch mir (.) ganz neue (.) Fähigkeiten aneignen musste weil ich da auch (.) saß und dachte okay ich habe von dieser Komponente noch ni:e etwas gehört" (Bm:627-635)*

Die Aussage *„Open Source äh (.) ab ins kalte Wasser"* zeigt noch einmal, dass Giovanni S. keine Erfahrungen bezüglich des Programmierens von Software besaß, als er angefangen hat, sich mit Open Source zu beschäftigen. Doch auch nachdem er sich schon mit verschiedenen Projekten wie Webdiensten und kleinen Zusatzprogrammen viel spezifisches Wissen und Können angeeignet hatte, reichten seine Fähigkeiten immer noch nicht vollständig für sein neuestes Projekt aus. Diese erneute *Diskrepanzerfahrung* diente wiederum als Motivation, sich das notwendige weitere Wissen anzueignen. Dies wird in den folgenden Zeilen aus dem obigen Zitat ersichtlich:

> *„ja zum Teil (.) sind die Fähigkeiten sicherlich die man da braucht überschneidend (.) aber zum Teil auch so: vö:llig unterschiedlich so dass ich zum Beispiel für das Projekt was du gefunden hast (.) auch mir (.) ganz neue (.) Fähigkeiten aneignen musste" (Bm:631-634)*

Wie gerade dargestellt lässt sich die zu Beginn der Beschäftigung mit FLOSS vorherrschende Lernmotivation gut anhand der subjektwissenschaftlichen Lerntheorie beschreiben. Wer noch

nichts oder wenig mit Informatik zu tun hatte und nicht programmieren kann, wird vor einem offenen Quellcode zunächst eine sehr starke *Diskrepanzerfahrung* machen. Seine eigenen Fähigkeiten erlauben es nicht, das Programm ohne weiteres zu verstehen oder gar sinnvolle Änderungen daran vorzunehmen. Wenn nun aber ein genügend starker Wille vorliegt, diese Diskrepanz zwischen aktuellem Vermögen und erwünschtem Vermögen (das Programm zu verstehen oder es den eigenen Wünschen entsprechend zu erweitern) zu überwinden, dann wird der Lernende geeignete Handlungen - das Erlernen der gewünschten Fähigkeiten beispielsweise - unternehmen, um die Diskrepanz zu überwinden.

Im Weiteren tritt jedoch bei den meisten Entwicklern bald eine neue Komponente hinzu, wenn die erste Frustration - ausgelöst durch die voraussetzungsreiche Komplexität des Programms und Programmiersprache - nachlässt. Dieser weitere Aspekt lässt sich vielleicht mit der Freude vergleichen, die manche Menschen empfinden, wenn sie logische Denkrätsel lösen. Diese beschreibt die *intrinsische Prozessmotivation*, welche die Freude an der Tätigkeit an sich als Motivation für Handlungen angibt. Gerhard P. sagt hierzu:

> *„Des sin hier gleich noch zwei weitere mit eingebaut (.) wenn ma die Technik einmal kann (.) //mhm mhm// muss mas natürlich gleich mal ausprobieren was noch geht" (Am:216-218)*

Hier wird die Freude am Ausprobieren und am Herumspielen mit den mühsam erarbeiteten Fähigkeiten deutlich. Die einmal erlernten Fähigkeiten lassen sich leicht für andere Zwecke und Kontexte nutzen und dieser Vorgang scheint einen intrinsischen Wert zu haben. Bei Giovanni S. kam dieses spielerische Element sogar schon früher zum Tragen:

> *„Im Vergleich zu so meinen Klassenkameraden überhaupt einen Computer unter meiner Kontrolle sag ich mal bekommen (.) ä:h und dann habe ich so einmal intensiv drei Jahre (.) fast jeden Abend gespielt (.) dann: (.) das war nicht direkt Open Source aber wie ich überhaupt zu diesen ganzen Thema Programmierung komme (.) dann habe ich ein eigenes Spiel entwickelt (.) hab festgestellt ist ja //mhm// ganz lustig" (Bm:575-580)*

Nachdem die ersten Schwierigkeiten mit der Computerbedienung gelöst waren, findet Giovanni S. Gefallen an den bereits vorhandenen Spielen und entwickelt aus der Freude am Spielen heraus den Wunsch, ein eigenes Spiel selbst zu gestalten und eignet sich hierfür die nötigen Programmierkenntnisse an.

5.2.3 Entwicklung von Software

> *„Und natürlich auch von den Quelltexten is dieses Zeug ja auch durch Versuch und Irrtum entstanden" (Am:503-504)*

Dieser Abschnitt soll darstellen, wie sich die Entwicklung der Software nun konkret realisiert, es wurde bereits mehrfach erwähnt, dass sich dieser Prozess nicht komplett unabhängig vom Prozess der Wissensaneignung darstellen lässt. Dies verdeutlicht auch das oben genannte Zitat auf prägnante Art und Weise. Der Quelltext ist durch „*Versuch und Irrtum"* entstanden. Irrtümer aus denen sich etwas lernen ließ. Doch jenseits dieses Aspekts gibt es noch einige andere relevante Blickwinkel auf die Entwicklung von Freier, bzw. Open Source Software, die zumindest in der Darstellung losgelöst sind von der Wissensaneignung des Entwicklers.

> *„Sondern die sin halt einfach auf einer Seite quasi gewachsen (.) des fing so als ganz kleine Tabelle an (.) halt immer mit nem Screenshot zwei drei Aufgaben (.) und des dann da so im Laufe der Zeit halt (3) explodiert" (Am:310-313)*

Hier zeigt sich auch ganz deutlich eine starke Software-Orientiertheit in den Ausdrücken „*die sin halt einfach auf einer Seite quasi gewachsen*" bzw. „*dann da so im Laufe der Zeit halt (.) explodiert*". Die ersten Beschreibungen stellen die Homepage sprachlich einem Lebewesen ähnlich dar, das *halt einfach* aus eigenem Antrieb *wächst*. Der Ausdruck „*halt explodiert"* impliziert eine dramatische **Eigendynamik**, die in der Beschreibung ohne einen weiteren Akteur auskommt. Dieses Moment der Eigendynamik lässt sich auch bei Dieter V. an folgender Stelle erahnen:

„das war so quasi der (.) initiale Punkt (.) wo ich zum ersten Mal ja ebenhalt (.) damit was gemacht hab und dann ist natürlich dann ganz viel weiteres mal dazu gekommen" (Dm:21-23)

Besonders auffällig ist hier das Wort *„natürlich",* hinter dem sich verschiedene Bedeutungen verbergen können. Es erscheint allerdings fast so, als ob auch hier eine Dynamik beschrieben wird, die über die willentliche Selbstbestimmung des Akteurs hinausgeht. Welche Elemente könnten dies konkret sein? Zum einen ist der Entwickler meist durch die üblichen Programmiersprachen beschränkt, sofern er nicht eine neue entwickelt. Was dies konkret bedeutet, beschreibt Gerhard P. wie folgt:

„und des is auch Zeug was der Browser jetzt erst neu unterstützt (.) letztendlich (.) //ja ja// also des is relati:v (.) des hät ma vor vier oder fünf Jahren (.) diese Sachen noch gar nicht machen können (.) da wärs ohne dieses Java Applet sowas gar nicht möglich gewesen (.) //mh mh// (2) also des is auch jetzt auch mit HTML fünf gibts ga:nz neue Möglichkeiten" (Am:721-725)

Doch auch Momente des Zufalls scheinen zu beeinflussen, wie die Software gestaltet wird:

„und in einem Programm wird halt mal des genutzt weil ich mich wieder dran erinnert hab das des da drin war rauskopiert (.) zusammenkopiert (.) und (.) läuft (.) gut (.) //mh mh// also des heißt des is teilweise nicht die optimale Lösung sondern halt eine die funktioniert und dann muss es auch wieder weitergehen" (Am:510-514)

Was in einem Kontext gelernt und erarbeitet wurde, kann oft auch für andere Programme genutzt werden. Diese Möglichkeit zur modularen Gestaltung kann daher auch ganz bewusst genutzt werden:

„Ah ja genau was noch dabei is (.) bei diesen Python-Programmen sind immer noch so eingebaute Infoseiten die man quasi unabhängig vom eigentlichen Programm einfach so als Steinbruch nutzen kann" (Am:249-252)

Im folgenden Zitat hat die Möglichkeit, ein Skript interaktiv zu gestalten, zu der kreativen Überlegung geführt, diese bereits vorhandene Funktionalität für ein Quiz zu nutzen:

> *„Und dann halt gemerkt oh man kann ja auch n Skript interaktiv gestalten (.) also (.) ma gibt ne Eingabe (.) es kommt ne andere Ausgabe je nachdem was ma eingibt (.) und dann halt überlegt naja macht ma halt mal n Quiz damit" (Am:36-39)*

Dies bedeutet nicht, dass allein schon die Möglichkeit, eine Veränderung durchführen zu können, als Handlungsauslöser fungiert:

> *„Und (.) ja ich habs jetzt mal hier Inkompetenzkompensationskompetenz ne (.) //@(.)@// also im Prinzip (.) Kompetenz is ja der neue Schlüsselbegriff (.) damit müss ma unsere Schüler auch ausstatten (.) also //@(.)@// eigentlich durch Training wie Tests gemacht werden schneiden die ja auch inhaltlich bei egal welchem Thema besser ab (.) //mh mh// also des wär son Nebeneffekt der damit schön zu fördern wäre (2) ja (.) naja wolln mal schaun (.) vielleicht bemerkt ja noch irgendjemand was damit möglich wäre" (Am:618-624)*

In diesem Fall war auch der konkrete Wunsch von Gerhard P., seine Schüler mit nützlichen *Kompetenzen auszustatten* handlungsfördernd. Man könnte hier eventuell von einer *internal self-concept-based motivation* sprechen. Bei einer genügend starken Motivation, die von den eigenen Überzeugungen herrührt, entsteht der Wunsch, diese selbst entwickelte Software auch anderen zu Verfügung zu stellen. Den Gründen und Motiven für die Veröffentlichung von FLOSS wird sich der nächste Abschnitt widmen.

5.2.4 Freigabe

Wurden die bereits vorgestellten Schritte erfolgreich durchlaufen und eine Software erstellt, dann bleibt noch die Frage zu klären, wie und warum es zur öffentlichen Freigabe der Software kam und wie die beteiligten Personen das Geschehen beschreiben. Die hierfür wohl am schwersten zu interpretierende und zugleich die ergiebigste Darstellung lieferte Eugen V.:

> *„Bislang wenn ich beteiligt war (.) gings also für mich entweder darum etwas für mich anzupassen und auch gleichzeitig das Ganze andern zur Verfügung zu stellen (.) weil es halt fertig war (.) und=und ich es immer schon blöd fand einfach Wissen irgendwo vergammeln zu lassen (.) deswegen hab ichs einfach Open Source gestellt=gestellt (.) irgendwo hochgeladen und fertig" (Cm:143-147)*

Die Darstellung, dass es darum ging etwas „*für mich anzupassen und auch gleichzeitig das Ganze andern zur Verfügung zu stellen"* erscheint auf den ersten Blick so einleuchtend, dass die darauf folgenden Erläuterungen im ersten Moment irritieren. „*weil es halt fertig war"* und „*ich es immer schon blöd fand einfach Wissen irgendwo vergammeln zu lassen"* sind nicht die Antworten, die man von einem sich prosozial verhaltenden Akteur erwarten würde. Um einen Vergleichshorizont aufzuzeigen hätte er sagen können, dass er es anderen zur Verfügung stellt, damit auch die es nutzen können oder weil er denkt, dass sie ein großes Interesse daran haben könnten. Stattdessen verharrt die Beschreibung wieder auf einer abstrakten, technischen Ebene. Das Programm war „*halt fertig"* und das Wissen sollte nicht „*vergammeln"*. Der letzte Hinweis „*irgendwo hochgeladen und fertig"* zeigt auch kein weiteres Interesse daran, ob es nun tatsächlich von anderen genutzt wird oder nicht. Das Interesse scheint beim zur Verfügung Stellen zu enden. Dies bestätigt auch eine weitere Textstelle, in der es mit dem Umgang um den Bachelorarbeit geht.

> *„Und sonst (.) ja auch jetzt zuletzt bei meinem bis:her größten Projekt (.) meiner Bachelorarbeit da ging es mir auch nicht (.) darum (.) das Ganze anderen vielleicht zur Verfügung zu stellen (.) weil es ist sehr spezifisches Wissen v=von*

dem auch (.) eh wahrscheinlich niemand (.) außer dem in (.) dem Lehrstuhl (.) wo ich geschrieben habe (.) ü=überhaupt etwas mit anfangen (.) können (.) aber (.) es ging halt (.) ja (.) sagen wa ihnen zur Verfügung zu stellen (.) ja=schon (.) aber es ist für mich keine (3) mh (2) keine Grund- moralische Einstellung (.) allen zur Verfügung zu stellen ich finde es einfach nur blöd wenn ich irgendwas mache und es vergammelt dann (.) und i=ich weiß ich werde damit nie=nie wieder etwas mit anfangen (.) zumindest ja in: fünf bis zehn Jahren (.) werde ich damit eh nichts anfangen und (.) in der (.) Technik ist es ja so (.) dass nach fünf bis zehn fünfzehn Jahre (.) wenn man irgendwas gemacht hat (.) fünfzehn Jahren (.) ist das eh wertlos (.) dann kann ich´s niemand mehr was damit anfangen" (Cm:147-160)

Er sagt hier explizit, dass es ihm nicht darum ging „*das Ganze anderen vielleicht zur Verfügung zu stellen (.) weil es ist sehr spezifisches Wissen"* mit dem wahrscheinlich niemand überhaupt etwas anfangen kann, nur um ein paar Wörter später zu sagen, dass es doch irgendwie um das zur Verfügung Stellen geht. Es gibt hier einen Konflikt, dessen er sich zumindest ansatzweise bewusst ist, aber Schwierigkeiten hat, ihn adäquat darzustellen. Möglicherweise liegt der Konflikt in der Sichtweise auf die Handlung des zur Verfügung Stellens. Wie bereits mehrfach erwähnt wird oft eine Software-zentrierte Sicht eingenommen. Es geht darum, dass die Software zur Verfügung gestellt wird, also prinzipiell nutzbar gemacht wird. Mit Blick auf den möglichen Nutzer reicht dies jedoch nicht aus, denn da geht es noch einen Schritt weiter mit der Frage, ob die Software tatsächlich von jemandem genutzt wird. Den Zusatz „*aber es ist für mich keine (3) mh (2) keine Grund- moralische Einstellung"* wählt Eugen V. möglicherweise, um sich von einer moralisch konnotierten, „etwas für andere tun"-Perspektive zu distanzieren. Doch warum ist es dann so wichtig, die Software zu veröffentlichen? Die in diesem Zusammenhang ins Auge fallende Metapher ist *„vergammeln"*. Dieser Begriff kommt eigentlich nicht aus dem Bereich der Technik, sondern aus dem Bereich des Organischen und wird insbesondere bei Lebensmitteln verwendet. Hier ist es jedoch das Wissen, das Gefahr läuft, zu vergammeln, also sich mit der Zeit zu zersetzen und nutzlos zu werden. Im Bereich der Computertechnik gibt es ständig Neuerungen und Umbrüche. Mit einem Computer von vor 10 Jahren kann heute kaum noch jemand etwas anfangen. Was für die Hardware gilt, gilt in gleicher Weise auch für die Software, die ständig angepasst werden muss. Mit einem Betriebssystem, das keine aktuelle

Treiberunterstützung mehr besitzt, kann man auf einem neueren Rechner nichts mehr anfangen. Es scheint also tatsächlich so zu sein, dass die Entwicklung rasant vonstatten geht und die Gefahr, dass etwas, in das man selbst viel Zeit und Mühe investiert hat, schnell nutzlos wird, ist real.

Um nun im Sprachbild des *„vergammelns"* zu bleiben, verhält sich Eugen V. wie jemand, der als Hobbygärtner Tomaten anbaut. Zur Erntezeit hat er dann auf einmal viel mehr Tomaten, als er selbst nutzen kann[26] und möchte nicht, dass etwas, in das er viel Zeit und Mühe investiert hat, einfach so vergammelt. Deswegen gibt er die überflüssigen Tomaten in einen Karton auf den er *zu verschenken* schreibt und stellt ihn auf die Straße *„und fertig"*. Ob die Tomaten dann tatsächlich von jemand abgeholt und gegessen werden, ist ihm dann egal.

Diese Einstellung mag auf einige Entwickler zutreffen, doch gibt es auch viele, die sich durchaus Gedanken darüber machen, was aus ihren „Tomaten" wird. Dieter V. sagte hierzu:

> *„Im Prinzip wenn man richtiges Open Source Projekt hätte ernsthaft draus machen wollen oder wenn ich das hätte machen wollen (.) dann ähm müsste ich ja Öffentlichkeitsarbeit und solche Geschichte machen also dann müsste ich ja wie es bei (.) proprietäre Software auch ist müsste ich das Produkt erst mal verkaufen (.) weil ne Software irgend einen Quelletext zu schreiben und irgend einen auf irgend einen @Webrechner@ dieser Welt hochzuladen (.) das interessiert keinen Menschen (.) also das ist immer das ist dann halt (.) das wäre dann schon große Glücksache wenn jetzt da jemand kommt und sagt okay (.) das möchte ich jetzt haben und weiter verwenden" (Dm:239-247)*

Bei Dieter V. ist ein ausgeprägtes Bewusstsein darüber vorhanden, dass, um Software für andere nutzbar zu machen, sie auch beworben werden muss. Eine Möglichkeit, diese Schwierigkeit zu überwinden, ist, seine Software oder oft auch kleine Verbesserungen über bekannte, bereits etablierte Projekte wie Linux zu verbreiten.

[26] der Aspekt, dass Software prinzipiell unendlich teilbar ist im Vergleich zu Tomaten wird hier vernachlässigt, da es mehr um die Motivation von Eugen V. geht.

„wenn man jetzt Erweiterungen schreibt ähm (.) dann ist jetzt gar nicht immer so ganz klar ob es ein Code ist der nur für einen selber zu gebrauchen ist (.) oder ist es ein Code (.) den auch andere Leute gebrauchen können (.) und Sachen oder eine Sache von der ich immer glaubte oder glaube oder inzwischen auch weiß (.) dass sie andere sie gut gebrauchen können (.) die hab ich dann an den Entwickler (.) von der Software wieder zurück gegeben und gesagt (.) hier guck mal (.) hab ich gebaut (.) willste haben (.) und dann hat er das eben eingebaut (.) und das ist jetzt heut zu Tage wenn man sich zum Beispiel Debian Linux installiert dann (.) ist es einer der ganz vielen Bestandteile die da mit drin ist (3) ja (2) Vorteile den ich davon hab (.) das es da drin ist (.) ist wenn es jetzt neuere Versionen von dieser Software gibt (.) ähm die ich ja sowie so einsetze (.) em das ich dann immer meine Erweiterung nicht wieder neu da reinbauen muss (.) sondern dass die einfach schon mit drin ist (2) und das ich mich darum nicht mehr kümmern muss" (Dm:129-141)

Dieser Schritt muss nicht zwangsläufig altruistisch motiviert sein, sondern kann durchaus auch zu konkreten Vorteilen für den Entwickler führen. So kann er, wie oben dargestellt, eine Arbeitserleichterung durch das Einbringen seiner Verbesserungen in das große Projekt haben. Er ist ja nicht nur Entwickler, sondern auch Nutzer. Auch die Möglichkeit, Geld zu verdienen, wird nicht ausgeschlossen:

„die GPL (.) also das ist ja eigentlich so ähm die ähm die Lizenz em die Open Source Software (.) eben halt em initiiert hat ähm (.) ähm die ähm (.) sieht ganz klar vor das man mit Open Source Software auch Geld verdienen kann (.) also das war schon immer so die Idee dabei (.) ähm man verdien- man verdient (.) das Geld halt nicht über die Lizenzen (.) ähm sondern man verdient das Geld dafür dadurch dass man einer ähm der wichtigsten Kenner der Software ist" (Dm:171-177)

Dieter V. weist auf die in einigen Lizenzen ausdrücklich eingeräumte Freiheit hin, auch Geld damit zu verdienen. Dies ist unter anderem als *„wichtigster Kenner der Software"* möglich, der von anderen für sein Wissen im Rahmen von Support, Schulungen oder neuen technischen

Lösungen bezahlt wird. Aus einer pragmatischen Perspektive heraus hat es sogar viele Vorteile, wenn sich damit auch Geld verdienen lässt oder sich auch andere Vorteile dadurch gewinnen lassen. Dadurch, dass die Projekte nicht nur auf die altruistische Motivation der Beteiligten setzen müssen, können sie viel größer werden und stabiler arbeiten, da ein jeder, der etwas dazu beiträgt, ja selbst einen Nutzen davon hat und deshalb sehr wahrscheinlich eine große Motivation hat, dies auch weiterhin zu tun. Im folgenden Abschnitt erläutert Dieter V. noch ein paar mögliche Vorteile, die durch das Veröffentlichen von Open Source Software zustande kommen können:

> *„Ich hab noch keine Idee ähm wie man (.) das jetzt ähm (.) in diesem reifen Zustand des Produktes irgend- irgendwie was daraus machen könnte das man es wirklich verkaufen könnte also (.) und wenn ich es nichts verkaufen kann em kann ich es auch halt unter einer Open Source Lizenz ähm veröffentlichen und hoffen das ich auf diese Art und Weise zumindest von irgendjemanden Feedback bekomme (.) oder irgendjemand ähm äh Fehler korrigiert em oder ich einfach einen Job bekommen dadurch das irgendjemand sieht ohh guck mal der kennt sich damit aus (.) ist ja auch immer Selbstwerbung wenn man sowas veröffentlicht" (Dm:229-236)*

Konkret genannt werden die möglichen Vorteile, Feedback zu bekommen und so etwas zu lernen, die Chance, dass jemand anders den selbst geschriebenen Programmteil optimiert und wiederum das Ergebnis zur Verfügung stellt, sowie Selbstwerbung, die auch zu Jobangeboten führen kann.
Zusammenfassend lässt sich sagen, dass die Freigabe selbst kaum Aufwand bedarf und es oft viele gute praktische Gründe dafür gibt. Möchte man jedoch ein eigenes Projekt starten und dies auch bekannt machen, muss viel investiert werden, wozu nicht jeder bereit ist:

> *„Aber mir fehlt bisher die Begeisterung für ein einzelnes Projekt (.) um mich da so stark zu involvieren" (Cm:250-251)*

Andere wie Gerhard P. möchten ihr technisches Wissen auch für andere nutzbar machen und suchen nach Möglichkeiten dies zu tun:

> *„Ähm dann auch viel so audio Dinger da hab ich mal überlegt so für die Sprachler wobei die warn da nicht so dran interessiert weils halt (.) ja (2) zu aufwendig" (Am:326-328)*

Oder an anderer Stelle: „*Vielleicht find ich ja da noch Nutzer dafür?" (Am:755).*

Doch die eigene Begeisterung muss nicht unbedingt von anderen geteilt werden und obwohl der eigene Nutzen und die Effizienz im Vordergrund stehen müssen noch andere Elemente mit ins Spiel kommen, die den Wunsch erzeugen, dass die eigene Software möglichst vielen Menschen bekannt wird.

> *„I: Gibts noch irgendwas was ihnen aufm Herze:n brennt oder erschwä:hnenswert erscheint oder?*
> *Am: ne was ((räuspern)) mich interessieren würde wär noch: (.) ähm obs ne Möglichkeit gibt über soziale Netze des stärker zu bewerben" (Am:805-807)*

Interessanterweise trat in jedem Interview eine starke Positionierung entweder gegenüber den FLOSS-Philosophien oder gegenüber den Nutzern proprietärer Software auf, die durchaus als idealtypisch zu sehen ist. Da es jedoch nicht möglich war, die Positionierung im zeitlichen Verlauf zu verorten, soll an dieser Stelle nur auf die Diskussion der Positionierung im nächsten Kapitel im Rahmen der Grounded Theory verwiesen werden.

Die in diesem Kapitel diskutierten Kategorien lassen sich als Veränderungen auf drei verschiedenen Ebenen darstellen: Die Auslöser und die Freigabe finden auf der Ebene der Umwelt, die Wissensaneignung auf der Ebene des Akteurs und die Entwicklung der Software auf der Ebene des Produkts statt. Diese drei Ebenen dienen als Ordnungsstruktur für die Grounded Theory, die im nächsten Kapitel ausgeführt wird.

6. Veränderung: Kernkategorie einer Grounded Theory

Dieses Kapitel möchte mit Hilfe einer Grounded Theory eine Antwort auf die Forschungsfrage **„Wie etabliert sich die Entwicklung von Freier und Open Source Software und was ist für sie charakteristisch?"** geben. Basierend auf den Überlegungen des vorangehenden Kapitels[27] lässt sich sagen, dass stetige Veränderung eine grundlegende Eigenschaft der digitalen Welt ist, die sich auch in den empirisch erhobenen Daten widerspiegelt. Der notwendige Lernprozess und die Entwicklung der Software selbst zeigen, dass substantielle Veränderungen sowohl auf der Ebene des Entwicklers, als auch auf der Ebene der Software notwendigerweise stattfinden müssen. Auch im Bereich der Motive für das zur Verfügung Stellen hat das Beispiel des *vergammelnden* Wissens deutlich gemacht, dass auch hier automatisch geschehende Veränderungen des Werts beziehungsweise des Nutzens der einmal erstellten Software eine zentrale Rolle spielen.

Das Element der **Veränderung** ließ sich ebenfalls in allen durch die verschiedenen Stufen des Kodierens entwickelten Kategorien wiederentdecken und wurde daher die Kernkategorie der hier entwickelten Grounded Theory. Als Antwort auf die Forschungsfrage lässt sich also feststellen: „**Die Entwicklung von FLOSS ist geprägt von substantiellen Veränderungsprozessen, die sowohl auf der Ebene des handelnden Individuums, als auch auf der Produktebene stattfinden und auf Veränderung in der Umwelt des Akteurs abzielen."** Im Folgenden soll nun die Grounded Theory anhand der drei gerade genannten Ebenen detailliert unter Rückbezug auf die erhobenen Interviews dargestellt werden.

[27]Die hier gewählte Darstellung der Grounded Theory ist zwar in sich geschlossen und verständlich, jedoch werden Aspekte, die bereits im Kapitel 5 erarbeitet wurden, nur kurz zusammengefasst wiedergegeben, um überflüssige Wiederholungen zu vermeiden.

6.1 Veränderungen auf der Ebene des Akteurs

Das erste und bereits ausführlich dargestellte Element der Veränderung auf der Ebene des Akteurs ist die **Wissensaneignung,** die sich als dynamischer Lernprozess darstellen lässt, der unterschiedliche Aspekte vereinen kann. Hierauf wurde bereits in Kapitel 5.2.2 unter Berücksichtigung verschiedener Lerntheorien intensiv eingegangen. An dieser Stelle soll nur nochmals zusammenfassend festgestellt werden, dass es sich hierbei eindeutig um ein Phänomen handelt, das notwendigerweise der Entwicklung von FLOSS vorausgeht oder gleichzeitig mit ihm erfolgt. Die Wissensaneignung ist eine notwendige Veränderung auf der Ebene des Akteurs, also des Entwicklers ohne die eine gelungene Softwareentwicklung nicht möglich wäre.

Die zweite Kategorie der **Positionierung** wurde bereits angesprochen und soll nun hier ausführlich besprochen werden. Zunächst vollkommen unerwartet hat sich in jedem Interview eine deutliche ideologische Positionierung gezeigt. Bei Gerhard P. zeigte sie sich in Form einer starken **Abgrenzung** gegenüber anderen Lehrern:

> *„Und dann halt auch das was (.) naja was machen die meisten Lehrer wenn sie irgendwie so (.) multimedial=modern sind (.) die klicken sich entweder äh des sind (.) dann schon die besten mit Mediator irgendwie son Mistding zusammen (.) ich mein da gibst ganz schöne Sachen im Netz aber das ist technisch katastrophal" (Am:50-54)*

> *„Oder halt die anderen oah Powerblöd (.) wenn ma ne Powerpoint macht dann ist man ganz vorne mit dabei (.) und des ist halt Quatsch" (Am:56-57)*

Hier wird ein stark abwertendes Vokabular wie „*Mistding"*, „*Powerblöd"* oder „*technisch katastrophal"* verwendet, um sich selbst und das eigene Handeln davon abzugrenzen und aufzuwerten. Möglicherweise wurde die Abgrenzung durch einen gewissen Rechtfertigungsdruck erzeugt, der durch eine ihn alltäglich umgebende, sich jedoch von seiner eigenen unterscheidende Praxis entsteht. Eugen V. zeigt hingegen eine Abgrenzung gegenüber der Philosophie von Open Source:

„I=ich nutze Software (.) also wo es (.) sag ich mal (.) geht und wo es sinnvoll ist (.) ähm die Open Source ist (.) aber (.) an für sich (.) also (.) Open Source ist für mich wie gesagt keine (.) Philosophie (.) keine Moralvorstellung (.) dess muss Open Source sein (.) dass ist eine von vielen Möglichkeiten (.) es gibt mh es gibt Projekte und bei denen macht es Sinn diese Projekte Open Source laufen zu lassen (.) das sind zum Beispiel (.) es ist alles Ehrenamtliche (.) alles was irgendwie ehrenamtlich läuft das sollte Open Source laufen denn (.) sobald es an einen Konzern gekoppelt ist (.) em geht die Kontrolle (.) des em der ganzen (.) Gemeinschaft (.) die d=dann die dann=davon Nutzen ähm trägt (.) die sich für sich entwickelt (.) geht die verloren (.) aber ich finde Open Source an sich (.) es ist also i= ich bin gegen (.) ähm (.) es gibt viele Open Source Verfechter (.) die wollen nur Open Source haben (.) da bin ich gegen" (Cm:195-206)

Er grenzt sich gegenüber einer „*Philosophie*", einer „*Moralvorstellung*" hinter Open Source ab, sowie gegenüber den vielen Open Source Verfechtern, die nur noch Open Source haben wollen. Es ist zu vermuten, dass die in Kapitel 2 bereits ausführlich dargestellten, deutlich ausgeprägten Positionen ideologischer Ausrichtung im Bereich der FLOSS zu dieser klaren Abgrenzung geführt haben. Wer sich etwas mit Freier oder Open Source Software beschäftigt, stößt schnell auf die dahinterliegende Ideologie und auf den Streit zwischen diesen Gruppen. Hieraus ergibt sich scheinbar die Notwendigkeit selbst Position zu beziehen. Dies zeigt sich auch in dem folgenden Zitat, in dem sich Giovanni S. über die Einstellung von Stallman, dem Gründer der Free Software Foundation, äußert:

„Selbst Ubuntu für zu radika- äh zu zu nah an proprietären Sachen hält aus dem ganz einfachen Grund (.) dort werden dann geschlossene Grafiktreiber verwendet und selbst das //mhm// ist dann eine Befleckung des des äh Geda- (.) des freien Gedankens er hat manchmal Recht ich sehe ihn son bisschen wie äh Alice Schwarzer in der Feminismus Bewegung die Frau kann auch fürchterlich nerven (.) manchma in manchen Dingen hat sie recht also das immer gut jemanden zu haben der auf den Tisch kloppen kann (.) der so ne radikale Position vertritt (.) wenn nach her die Mitte bei raus kommt ist das schon ganz gut" (Bm:116-125)

Besonders interessant ist hier zunächst die Metapher *„Befleckung des freien Gedankens",* die wiederum eine eindeutige moralische Konnotation besitzt. Hier der reine *„freie Gedanke"* der Open-Bewegung im ständigen Kampf gegen die böse, proprietäre Software. Jede Abweichung vom Ideal des „*freien Gedankens"* wie die nicht öffentlich zugänglichen Treiber in Ubuntu sind dann automatisch ein Zugeständnis an das „*Böse",* eine „*Befleckung des freien Gedankens".* Gegenüber dieser schwarz-weiß gefärbten Sichtweise nimmt Giovanni S. eine moderate Haltung ein und vergleicht Stallman mit Alice Schwarzer, die manchmal mit ihrer radikalen Einstellung „*fürchterlich nervt"* und doch einen wichtigen Beitrag für die Gesamtausrichtung der Bevölkerung liefert.

Die zweite Form der klaren Positionierung ist die **Identifikation** mit einem Gedanken oder einer Ideologie, wie sie ebenfalls bei Giovanni S. zu beobachten ist:

> *„Hab ich mich da mal umgescha:u:t (.) und dann gefiel mir die Ideologie dahinter (.) weil letztlich ist es schon ne ge- Art (.) von Ideologie (.) zu sagen (.) ok das ist freie Software jeder kann es ändern jeder kann äh dahinter ja die Sachen modifizieren (.) ka:nn: (.) än-ä:h kann eigene Features hinzufügen und es ist eben auch alles kostenlos" (Bm:16-19)*

Das obige Zitat steht gleich zu Beginn des Interviews, in der Passage, in der er erklärt, wie er dazu kam, das erste Mal FLOSS-Software zu verwenden. Dies zeigt, dass bei ihm die Entscheidung für die Verwendung von Freier Software nicht nur aus konkreten Notwendigkeiten entstand, sondern auch wegen der dahinter stehenden Ideologie, mit der er sich noch immer identifizieren kann:

> *„Meine Ideale sind letztlich eine äh (.) freie und transparente äh (.) oder eine freie Gesellschaft (.) ähm (.) und (4) und da passt Open Source nun mal wunderbar mit rein (.) in diese dieses Ideal (2) eine freie und (.) äh ach Gott (.) wie man auch immer es ausdrücken mag (.) freie transparente gerechte Gesellschaft" (Bm:525-528)*

Der Punkt der Philosophie hinter den FLOSS-Produkten wird noch im Kapitel 6.3 eine Rolle bei der Analyse der persönlichen Ziele spielen.

6.2 Veränderungen auf der Ebene des Produkts

> *„Das war viel weniger der äh das wu- war bei mir weniger der Umstand (.) dass ich mich dafür engagieren wollte (.) sondern das war einfach der Umstand dass ich mich dafür engagieren konnte (.) weil es halt Open Source war" (Cm:52-54)*

Diese Stelle zeigt bereits deutlich auf, worum es in diesem Unterkapitel gehen soll. Es geht nicht nur um die triviale Feststellung, dass Software während ihrer Entwicklung stark verändert wird, sondern dass FLOSS-Software dadurch, dass sie für jeden veränderbar ist, erst die Möglichkeit schafft, sich in diesem Bereich zu engagieren. Dies zeigt sich auch deutlich in weiteren Zitaten:

> *„Ich ähm ändere gerne Kleinigkeiten (.) wenn ich irgendwelche Software nutze (.) wenn ich merke (.) das könnte man besser machen (.) dann mach ich" (Cm:241-243)*

> *"Das ist eine ganz schicke Sache (.) was diese Technik erlaubt" (Am:118-119)*

Im ersten Zitat wurde keine direkte Notwendigkeit als Auslöser für die Verbesserung angegeben, sondern nur, dass die Software „verbesserungsfähig" ist. Das zweite Zitat personalisiert sogar die Technik in der Formulierung, dass sie „*eine ganz schicke Sache erlaubt*", es schreibt ihr also eine passive Rolle zu. Dieses Phänomen wurde unter der Kategorie **Modifizierbarkeit** in die Theorie aufgenommen. Hierzu wurde auch bereits im Unterkapitel 5.2.3 einiges gesagt. Ebenso zeigt sich im bereits erläuterten Aspekt des „vergammelnden Wissens", der in die abstrakte Kategorie **Nutzbarkeitsverlust** aufgenommen wurde, eine stetige Veränderung des Werts bzw. des möglichen Nutzens von Software.

6.3 Veränderungen auf der Ebene der Umwelt

> *„Und das ist eigentlich <u>das</u> was ich mir halt selbst immer gewünscht hätte für Arbeitsblätter (.) warum gibts das nicht? oder unglaublich aufwendig" (Am:144-146)*

Der in diesem Abschnitt eingenommene Blick ist in gewisser Weise eine Ergänzung zur Betrachtung der Notwendigkeiten als Auslöser in 5.2.1. Man könnte das Zitat als eine Notwendigkeit nach besseren Arbeitsblättern interpretieren und hätte damit sicher nicht Unrecht, aber das würde einen positiven Blick auf den dargestellten Sachverhalt verstellen. Eine andere Betrachtungsweise ist vielleicht durch das Verb „wünschen" zu rechtfertigen. Der Wunsch nach etwas, das es „nicht gibt" oder nur „unglaublich aufwendig" ist, hat ein kreatives, gestalterisches Moment in sich. Dieser Wunsch ist ein Wunsch nach einer konkreten Veränderung im direkten Lebensumfeld, zu dem hier auch die technischen Rahmenbedingungen einer Person zählen sollen. Im ersten geht es also um eine Nutzbarmachung bestimmter technischer oder zeitlicher Ressourcen, die eng mit einem gewissen Effizienzverständnis verknüpft sind. Dies zeigt sich in folgenden Zitaten von Gerhard P. ganz deutlich:

> *„Also so ganz banale simple Sachen (.) die aber sonst unheimlich Zeit kosten" (Am:325-326)*

> *„Ja und eigentlich durch diese ganz banalen Techniken die also keine große Geheimwissenschaft sind kann ma damit eigentlich schöne Sache machen" (Am:191-192)*

> *„Also das zum Beispiel auch n hübsches Ding und sowas hat ma an einem Nachmittag gemacht (.) und des is unglaublich nützlich" (Am:203-205)*

> *„Also solche kleinen Hilfsmittel" (Am:223)*

> *„Ja so=so ganz kleine Helferlein für n Alltag halt" (Am:799-800)*

Allein schon die Tatsache, dass an so vielen Stellen Bezug darauf genommen wird, zeigt auf, dass es eine gewisse Wichtigkeit für Gerhard P. besitzt. Das erste Zitat lässt sich klar dem Effizienzgedanken zuordnen. Das zweite und dritte Zitat verdeutlichen, das es nicht nur um die **Effizienz** geht, sondern auch um eine gewisse **Ästhetik**. Begriffe wie *„schön"* und „*hübsch"* fielen oft während des Interviews. Die letzten beiden Zitate weisen durch die Begriffe „*Hilfsmittel"* und *„Helferlein"* ebenso auf das **Gestalten** als Kategorie hin. Die Software ist ein Werkzeug, um den Alltag effektiver und schöner zu gestalten. Doch Werkzeuge können nicht nur dazu verwendet werden, um den eigenen Alltag zu verbessern, sondern eignen sich prinzipiell für jedes Ziel, das man verfolgt, auch für die ganz großen Dinge wie *Welt gestalten* und *Gerechtigkeit*. Im Falle der Open Source Software ergibt sich sogar die Möglichkeit, dass das Werkzeug nicht nur Mittel zum Zweck bleibt:

> *„Mein per- denke ich doch der größte persönliche Treiber hinter alle dem (.) dass ich Wel- die Welt da draußen gewissermaßen verändern möchte (.) und (2) ja (.) da passt dieses (.) diese die Möglichkeiten der Anpassung von Open Source Software natürlich super zu weil (.) ich brauch ein Werkzeug einfach mit dem ich ein (.) irgendetwas verändern möchte //ein Werkzeug// is für mich ja stimmt also fü- äh der Begriff Werkzeug ist gar nicht so falsch das ist für mich tatsächlich ein Werkzeug u:m (.) u:m (.) um etwas zu erreichen (.) was ich anders nicht erreichen könnte und ich natürlich wenn ich ein Werkzeug verwende äh möchte ich mich auch mit dem (.) äh äh (.) mit dem: (2) was dieses Werkzeug bedeutet äh (.) irgendwie identifizieren können (.) und das tue ich (.) im Fall von Open Source" (Bm:265-275)*

> *„Die weil die Idee da hinter ist (.) <u>super</u> //mhm// also d-(.) die Idee hinter Open Source ist ja (.) irgendwo eine Bereitstellung von:: (.) von:: (2) Software von (.) <u>Ressourcen</u> das geht ja durchaus in andere Bereiche rein (.) also Op-open Source ist <u>Teil</u> einer <u>größeren</u> Bewegung dieser ganzen Open Bewegung ähm die <u>Welt irgendwo</u> auch um es ganz <u>hoch</u> zu formulieren ein bisschen gerechter zu machen (.) indem man eben allen Menschen (.) <u>di:e</u> die Möglichkeit gibt Bildung zu erfahren und eben auch (.) zum Beispiel jetzt speziell Open Source (.) Software zu*

bekommen die genauso gut ist wie di:e (.) wie welche die man kaufen muss (.) so dass man irgendwie so ein Gerechtigkeitsaspekt äh da sichtlich mit drin hat (.) und meine @persönliche Motivation@ is auch einfach (.) wenn ich die Welt gestal-ten (.) möchte (.) also wenn ich das da draußen gestalten möchte (.) und das insbesondere im IT Bereich machen möchte (.) dann kann ich nicht irgendwelche geschlossenen Systeme nehmen //mhm// da kann ich nichts gestalten" (Bm:86-99)

7. Zusammenfassung und Ausblick

Die vorliegende Arbeit hat versucht, eine Antwort auf die Frage zu geben, wie sich die Entwicklung von FLOSS etabliert und was für sie charakteristisch ist. Hierzu wurde zunächst eine typische FLOSS-Entwicklerkarriere vorgestellt und analysiert. Aus den sich daraus ergebenden Aspekten und unter Zuhilfenahme aller Interviews wurde eine idealtypische FLOSS-Entwicklungsgeschichte erstellt. Diese Überlegungen dienten dazu, eine Grounded Theory über die Entwicklung von FLOSS einzuführen und anhand der Kernkategorie Veränderung darzustellen. Hierbei wurden alle relevanten Ebenen berücksichtigt und es erfolgte eine explizite Darstellung der Ebenen des Akteurs, der Software sowie der Umwelt. Die vorliegende Studie konnte sowohl die Etablierung der Praxis der FLOSS-Entwicklung in ihrem Verlauf nachzeichnen, als auch einige Antworten zu den Auslösern und Motivationen geben.

In zukünftigen Studien kann eine genaue Unterscheidung von Freier Software gegenüber von Open Source Software sinnvoll sein, wenn es um eine genaue Unterscheidung der Motivationen hinter dem Erstellen der Software geht. Auch eine systematische Untersuchung aller Faktoren unter Zuhilfenahme der Unterscheidung zwischen intrinsischen und extrinsischen Motivationen (Barbuto et al. 1998) könnte hierzu dienlich sein. Die Software-zentrierten Aspekte lassen sich möglicherweise gut in den theoretischen Rahmen der extrinsischen Motivationen einfügen.
Einige der Aspekte der intrinsischen Prozessmotivation erinnern an die Freude beim Spielen. Auch hier könnte eine weitere Untersuchung vielversprechend sein, um mögliche Verbindungen zur *homo ludens* Theorie (Huizinga 1938) bzw. der Ludologie[28] (Caillois 2001) zu entdecken.

Abschließend lässt sich feststellen, dass es sich bei der Entwicklung von FLOSS, um ein vielschichtiges, interessantes Forschungsfeld handelt, dass sicherlich von weiteren Studien unter anderen Blickwinkeln profitieren würde.

[28] Nach der Ludologie ist die Freude am Spiel angeboren. Das Individuum lernt durch das Überwinden von Hindernissen, indem es sich mit dem Problem beschäftigt.

A. Literaturverzeichnis

Barbuto, John E., Jr.; Sholl, Richard W. (1998): Motivation Sources Inventory: Development and validation of new scales to measure an integrative taxonomy of motivation. In: Psychological Reports 82, S. 1011-1022.

Benkler, Yochai & Nissenbaum, Hellen (2006): Common-based Peer Production and Virtue. The Journal of Political Philosophy 14 (4), S. 394-419.

Bundesministerium für Familie, Senioren, Frauen und Jugend (2010): Monitor Engagement (Nr. 2) – Freiwilliges Engagement in Deutschland 1999 – 2004 – 2009. Berlin

Caillois, Roger (2001): Man, Play and Games. University of Illinois Press, Urbana and Chicago.

Huizinga, Johan (1938): Homo Ludens. Rowohlt Taschenbuch Verlag, Hamburg, 21. Auflage, 2009

Holzkamp, Klaus (1995): Lernen-Subjektwissenschaftliche Grundlegung. Campus Verlag, Frankfurt/New York, Studienausgabe 1995

Kishida, Kouichi & Ye, Yunwen (2003): Toward an Understanding of the Motivation of Open Source Software Developers. 2003 International Conference on Software Engeneering (ICSE2003), Portland.

Newen, Albert & Bartels, Andreas (2011). Das Verhältnis von Mensch und Tier. Spektrum der Wissenschaften, April 2011: 283–308.

Perens, Bruce (1998): Open Source Definition.

http://opensource.org/docs/osd, Internetquelle (Stand 24.09.2013)

Przyborski, Aglaja & Wohlrab-Sahr, Monika (2009): Qualitative Sozialforschung. Ein Arbeitsbuch. Oldenbourg, München.

Schürmann, Lisa Kathrin (2013): Motivation und Anerkennung im freiwilligen Engagement. Kampagnen und ihre Umsetzung im Internet und Social Media. Springer VS, Springer Fachmedien Wiesbaden

Stallman, Richard (2013): Free Software Definition.

http://www.gnu.org/philosophy/free-sw.html, Internetquelle (Stand 31.08.2013)

Stern-Test: Wikipedia schlägt Brockhaus. 5.12.2007.

http://www.stern.de/digital/online/stern-test-wikipedia-schlaegt-brockhaus-604423.html, Internetquelle (Stand 25.09.2013)

Strauss, Anselm & Corbin, Juliet (1996). Grounded Theory: Grundlagen Qualitativer Sozialforschung. Beltz/PVU, Weinheim.

Tomasello, Michael (2010). Warum wir kooperieren. Suhrkamp, Berlin.

Weiner, Bernard (1994): Motivationspsychologie. Beltz/PVU, Weinheim, 3. Auflage, 1994.