Frei wie in Rede!

Betrachtung von Freier Software unter arbeitswissenschaftlicher Sicht

G. Glawe 2013/2014

Abstract

Open Source Software ist jedermann ein Begriff - zumindest denken viele zu wissen, was der Begriff meint. Aber wo kommt er her, was bedeutet er wirklich?
Richard Stallmann gründete 1985 - lange bevor das Internet groß wurde - die Free Software Foundation, die FSF. Als durchaus politische Aussage für das Recht auf freie Software (wie auch der Slogan „Protect Your Freedom — Fight „Look And Feel“ zeigt) begonnen, ist in der Geschichte der FSF mit dem GNU Unix “Linux“ nicht nur eins der besten kompletten Softwaresysteme überhaupt entstanden, es wurden auch Arten der Kooperation gefördert, die bis heute für viele Menschen undenkbar sind.

Hier soll Open Source Software unter arbeitswissenschaftlichen Gesichtspunkten untersucht werden. Welche Arbeitsformen entstanden durch die notwendige Zusammenarbeit von weltweit verteilten Programmierern und anderen Knowledge Workern? Wie wird eine solche Arbeit gesteuert, wird sie überhaupt gesteuert?

Dabei werden besonders die folgenden Aspekte betrachtet:
Ist Mitarbeit an OS Projekten Arbeit?
Werden andere Organisationsformen verwendet als in kommerzieller Produktion von Software?
Was ist das Menschenbild der Initiatoren von OS?
Welche Motivation gibt es, unbezahlt an OS Projekten mitzuarbeiten?

Dabei wird der Bezug zum Fordismus, Taylorismus, Toyotaismus sowie der Sozio-Technischen System von Trist gemäß der Arbeit im Kurs „Arbeitswissenschaftliche Ansätze - Rationalisierung und Humanisierung“ gezogen.

1 Einleitung und Hintergrund

1.1 *Einleitung und Hintergrund*

Im Kurs „Arbeitswissenschaftliche Ansätze - Rationalisierung und Humanisierung“ wird die Geschichte der Arbeit(swissenschaft) primär im Fokus der Produktion dargestellt. Im letzten Viertel des 20. Jahrhunderts kam mit dem Aufkommen der IT ein weiterer Handlungsstrang dazu: Die Geschichte der Knowledge Worker, speziell ab ca. 1985 auch „Freie Software“. In den Diskussionen des Kurses „Arbeitswissenschaftliche Ansätze“ wurde klar, dass im Themenkontext freier Software die Historie sowie die möglichen Auswirkungen auf die heutige Arbeitswelt des Knowledge Workers oft noch unbekannt sind.

Da ich selber - Abitur 1986 und Diplomarbeit 1992 - mit dem Entstehen des World Wide Webs in das Berufsleben eingestiegen bin und mich dabei freie Software auf verschiedene Arten begleitet hat, möchte ich mit diesem Paper das Thema Freie Software unter verschiedenen Gesichtspunkten näher betrachten.

In der Einführungswoche zum Studiengang Arbeitswissenschaften wurde zu Anfang der Taylorismus und Fordismus betrachtet. Zu Ende des historischen Wegs wurden partizipative Managementmethoden betrachtet. Die Gruppenarbeit in verschiedenen Formen veränderte das Bild der Arbeit. Die theoretischen Grundlagen dieser Gruppenform lieferte vor allem das Konzept des Sozio-Technischen Systems, welches durch die Untersuchung des Tavistock Institutes um 1950 hervorging (Siehe dazu Sturm 2011).

Dieser Ansatz wurde im Toyotismus mit dem vorgenannten Konzepten verbunden.

Aktuell setzt sich das Lean-Management-Konzept durch, welches Elemente des Taylorismus, des Floristischen Ansatzes und des Toyotismus enthält und verbindet.

Die Arbeitsweisen der hier betrachteten Open Source Projekte sind wieder etwas anderes. Eine dezentrale Arbeitsweise, die ohne das Internet schwer denkbar ist, führt zu neuen Gesetzen der Zusammenarbeit, des Menschenbildes und entwickelt - wie auch die Paradigmen der partizipativen Managementmethoden - einen Gegenentwurf zu dem tayloristischen Konzept der strikten Trennung von Konzeption und Ausführung der Arbeit.

Es gibt viele Einflussfaktoren für die Arbeitsformen in Gruppen. Zusätzlich zu den Faktoren derssoziotechnischen Systems gibt es technische, soziale, materielle und fachliche Einflüsse. Nicht zuletzt spielt die persönliche Konstellation eine Rolle.

In dieser Arbeit wird untersucht, welche Faktoren bei der Arbeit in Open Source Projekten eine Rolle spielen. Meine These ist, dass der Management Ansatz und die Arbeitsweisen der Gruppen bei technischer Führung viel mehr durch das Menschenbild der Führungsperson selber beeinflusst wird als durch wissenschaftliche Vorgehensweise. Daher stelle ich zuerst kurz die Theory X/ Theory Y vor, die einen ähnlichen Ansatz hat.

Die Motivation, die Arbeitsweise von erfolgreichen Projekten zu betrachten, ensteht aus der praxisorientierten Fragestellung heraus, ob sich die „Erfolgsfaktoren“ auch auf andere Projekte anwenden lassen. Damit verbunden ist die Frage - ist das Mitwirken in OS Projekten Arbeit? Oder ist es wie Linux Torvalds sein Buch nannte „Just for fun“?

1.2 *Freie Software und ich*

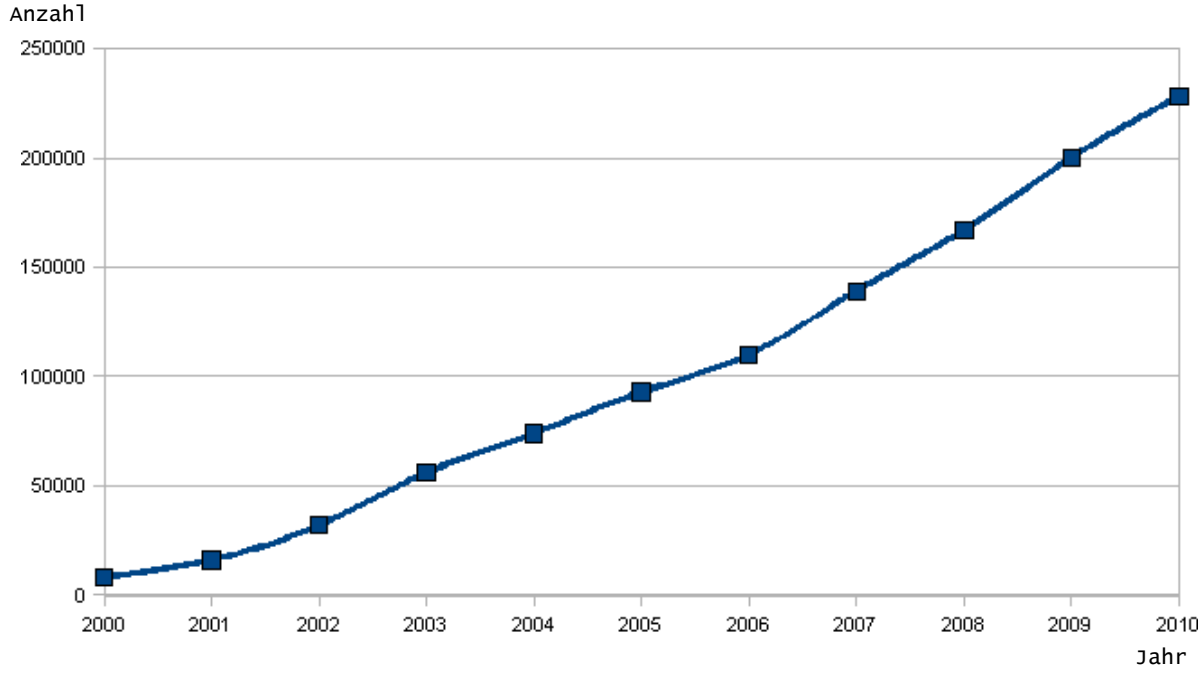
Wie kam ich selber zur freie Software? Die Geschichte spielte sich so oder so ähnlich ab:

1991 - sechs Jahre nach der Gründung der FSF (Free Software Foundation). In der Zwischenzeit hatte sich - besonders im universitären Umfeld - der Einsatz von freier Software etabliert. Ich selbst habe 1991 mein Diplomarbeit *vollständig* mit Hilfe von freier Software erstellt. Der Compiler[1] war der Compiler des FSF-Gnu Projektes, der GCC (Gnu Compiler Collection, ursprünglich Gnu C Compiler). Geschrieben habe ich die Arbeit - also die Dokumentation - mit Hilfe des Satzprogrammes TeX/LaTax, welches bis in die heutige Zeit als Satzprogramm im Layout der produzierten Darstellungen den meisten Office Programmen überlegen ist.[2]

Ein besonderes Abenteuer war es für mich damals, im Rechenzentrum der Technischen Universität Braunschweig auf 5,1/4“ Floppy Disketten ein komplettes Betriebssystem herunterzuladen. Das Internet, so wie wir es heute kennen, war in den ersten Anfängen. Für die Verbreitung von freier Software war die Möglichkeit des Internets, der Verteilung in die ganze Welt, ohne auf Mailboxverbindungen angewiesen zu sein, ein Kickstarter. Dieses Betriebssystem auf ca. 20 Disketten war natürlich Linux in der Version 0.96. Und mit dem Betriebssystem war der gesamte Quellcode und damit das gesamte Wissen über die Erstellung der Programme, also dieses Betriebssytems, zu bekommen. Für Studenten der Informatik und andere Wissenshungrige im Bereich Softwareentwicklung war das eine Goldgrube - das „geheime Wissen“ über Betriebssysteme, Hardwaretreiber und vieles mehr - vollkommen offen und frei verfügbar.

1.3 *Leitfaden*

Um eine Idee von der Anzahl der OS Projekte zu bekommen, hier eine Grafik einer Website, die Projekte beheimatet, nämlich SourceForge. Dort sieht man auf der Grafik die Entwicklung der Anzahl der dort beheimateten Projekte.



(Quelle: Wikipedia)

Aus der Vielzahl Themen der Open Source Welt (OS) habe ich vier herausgesucht, die verschiedene wichtige Meilensteine der Evolution von OS kennzeichnen. Dies sind:

1. Die FSF - Free Software Foundation mit der GNU Software,
2. Linus Torvalds mit der Linux Software,
3. das Paper „Die Kathedrale und der Basar“ von Eric Raymond und
4. die ASF Apache Software Foundation mit dem Apache Projekt.

Die Themen bilden folgende Meilensteine ab:

Mit der FSF fing der Gedanke einer freien Software an, zumindest in der Informatik-Szene mehr Beachtung zu finden.

Das bekannte Betriebssystem Linux war eines der ersten großen einflussreichen, wenn nicht das immer noch einflussreichste OS Programm. Es konkurriert direkt mit dem Betriebssystem Windows von Microsoft. Hier hat sich schnell eine Gruppe zusammengefunden.

Mit der Kathedrale und der Basar wurden die Erfolgsfaktoren von OS Projekten zusammengefasst und publiziert. Damit wurden die Ideen und Konzepte einer breiten Masse zugänglicher gemacht.

Die ASF stellt einen weiteren Evolutionsschritt dar. Sie ist kein einzelnes

Projekt, sondern eine Sammlung von Projekten. Die ASF hat für die Zusammenarbeit mehr formulierte Regeln, wie später noch ausgeführt wird.

Der Weg durch die Historie wird hier nach der Vorstellung der Theorie Y mit einem Begründer des Gedankens der Wissensfreiheit in der Software begonnen, Richard Stallman. Er war zunächst alleine auftreten und definierte moralische und politische Grundsätze, mit denen er seine Ansicht über Freiheit ausdrückte.

10 Jahre später hatte das wohl bekannteste Projekt den offiziellen Produktionsstart mit der Version 1.0: Linux.

Ein weiterer Meilenstein ist die Analyse von OS Projekten in „Die Kathedrale und der Basar“.

Zum Schluss betrachte ich noch kurz eine Organisation, die den Zusammenschluss vieler Projekte durchführt: Die Apache Software Foundation.

---

[1] Das Programm, welches aus Text ein ausführbares Programm macht.

[2] TeX beherrscht die Darstellung nach dem goldenen Schnitt, welchen man sich selbst im aktuellstem Word 2010 manuell erstellen muss.

1.4 *Arbeit*

1.4.1 *Was ist Arbeit*

Was ist Arbeit und wodurch zeichnet sie sich aus?

Als Einstieg ist dazu ein Bild zu sehen, welches die Antworten einer Gruppe von Studenten der Arbeitswissenschaft in dem Einstiegskurs 2013 auf die Frage „Was verbinde ich mit dem Begriff Arbeit?“ zeigt.

Luczak (1993) unterscheidet zwei Aspekte von Arbeit: Arbeit als Anstrengung und Arbeit als Produktion von Gütern oder Dienstleitung.

In der 1993 Ausgabe wird im Kapitel 1.1 gesagt „Arbeit ist somit eine besondere Form des Tätigseins neben anderen wie Spiel, Sport oder Lernen“

Zum Faktor Bezahlung zeigt die Tatsache, dass nicht bezahlte Arbeit extra als *unbezahlte* Arbeit benannt wird, dass normalerweise Arbeit bezahlt wird.

Daher wird die Fragestellung, ob Mitarbeit an OS Projekten tatsächlich Arbeit oder Hobby ist, schwierig. Es werden Güter produziert, die aber nicht verkauft

werden *dürfen*. Gemeinhin wird angenommen, dass die Mitarbeit an OS Projekten unbezahlt ist. Hierfür gibt es Gegenbeispiele im Abschnitt 1.6.2 - Bezahlte Mitarbeit an Open Source Projekten.

Die Annahme, dass Open Source, also freie Software identisch mit Free Software, also unbezahlte Software ist, ist ebenso falsch. Free ist ursprünglich im Sinne von ungebunden und frei zur Weitergabe gemeint, nicht kostenfrei. Dies spiegelt sich auch in der Entwicklung der verschiedeneren Lizenzmodelle wider. Neuere OS Lizenzen erlauben die Verwendung von Anteilen freier Software in kommerziellen Produkten durchaus.

Die Mitarbeit an OS Projekten hat viele Aspekte von Spiel, Sport und Lernen. Damit greift die Abgrenzung der Arbeit von Luczak nicht. Spielerische Aspekte zeigen sich z.B. in der Vielzahl kleiner Projekte, die spielerisch Themen aufgreifen. Es gibt Projekte, die selber Spielesoftware erstellen. Sportlichen Wettkampf gibt es bei konkurrierenden Projekten zum selben Thema.

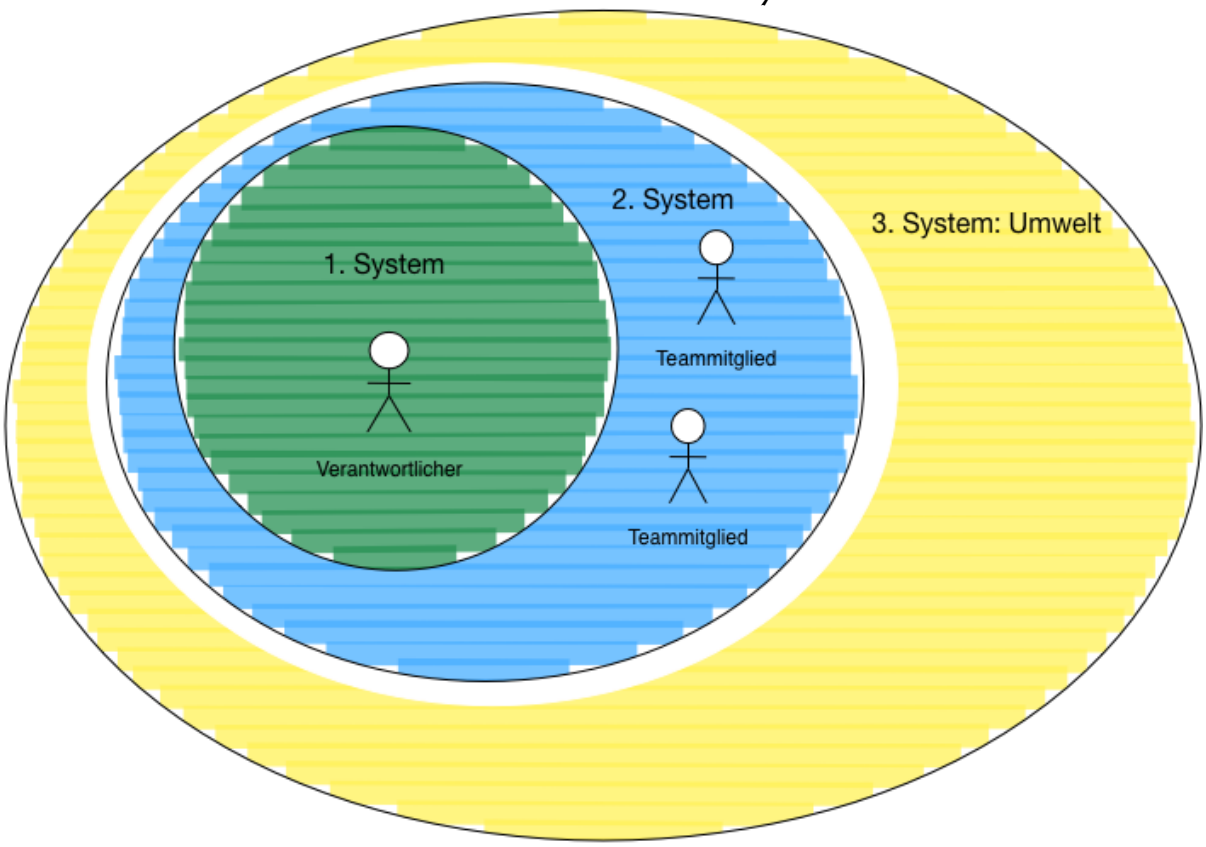
Und Lernen ist einer der Hauptmotivationsfaktoren bei der Mitarbeit oder der Beschäftigung an und mit OS Projekten. Wie kann man dabei lernen?

Normalerweise ist ein Programm eine sogenannte Black Box. Man kann nicht hineinsehen. Ist der Quellcode, also die Bauanleitung offen und frei verfügbar, kann direkt in den Bauplan hineingesehen werden. Und dort sind auch in Kommentaren die Hintergründe von Entscheidungen, wie etwas gebaut wurde sichtbar. Antworten auf die Fragen: „Wie hat der Programmierer das gebaut?“ oder „Warum hat er es so gebaut?“ sind direkt im Quellcode zu finden.

Um etwas ganz genau zu lernen, kann in den offenen Diskussionsforen im Internet der Entscheidungsweg nachvollzogen werden. Aktive Partizipation ist in den Foren durch eigene Fragen möglich.

1.5 *Organisationstheorien*

1.5.1 *Die betrachteten Systeme*



Wir betrachten bei der Organisation hier drei soziale Systeme: Die eigene Sicht, also der Manager oder der Vorgesetzte oder der Einzelarbeiter. Das zweite System beinhaltet die Teammitglieder und den Verantwortlichen. Das dritte System ist die Umwelt und beinhaltet die anderen beiden Systeme nicht.

1.5.2 *Der Mensch im Unternehmen*

Der Begründer des „Scientific Managements“ Frederick W. Taylor (1856-1915) sah den Antrieb zur Arbeit primär im Lohn[1], nicht bei einer in den meisten Menschen vorhandenen inneren Motivation. Seine Sicht auf die Arbeiter war es, dass sie Ausführende von Anweisungen sein sollten und dass das Wissen über die Arbeit bei wenigen gut ausgebildeten Menschen - meist Ingenieuren - zu finden sei. Das Umfeld für diese Ideen war in der Produktion zu finden.

Gruppen von Knowledge Workern gab es damals noch nicht in der Form wie heute.

Im Vordergrund stehen bei der Berufung auf Taylor meist eher die kleinteilige spezialisierte Arbeit, seine Sicht auf die Menschen wird dabei oft vergessen.

Natürlich sollte man auch das dritte System, die Umwelt nicht außer Acht lassen. Taylor betrachtete viele Produktionsschritte, die manuelle Arbeiten erforderten. Der manuelle Anteil der Arbeit ist seitdem in den Industrienationen stark zurückgegangen.

Interessant ist dabei, dass noch heute die tayloristischen Prinzipen nicht nur in Fertigungsbetrieben mit hoch standardisierten Produkten Anwendung finden, sondern auch in der Bearbeitung von Dokumenten und betriebswirtschaftlichen Abläufen wie der Auftragsbearbeitung.

Douglas McGregor entwickelte 1960 in seinem Buch „The Human Side of Enterprise“ eine humanistische Theorie der Arbeitsmotivation (MCGregor 2005). Dabei nahm er zuerst den tayloristischen Ansatz auf, um dann zu beschreiben, dass die Produktivität mit einem humanistischen Ansatz besser sei.

Dabei fragte er sich zuerst: „Was sind die Annahmen, die der Manager über seine Untergebenen trifft?“

Während bei Taylor also die möglichst perfekte Arbeitsanweisung für den Arbeiter das optimale Produktionsergebnis liefern soll, werden nun die Ansichten des Managers über die Arbeiter einbezogen.

Über diese Annahmen, die der Manager über die Arbeiter trifft ,wird auch die Arbeitsleistung des Arbeiters selber (System 2) verändert.

Gregor beobachtet folgendes:

Ein *Theorie X* Manager hat folgende Annahme über die Mitarbeiter:

Mitarbeiter sind Kosten, die überwacht und kontrolliert werden müssen. Arbeiter mögen es nicht, zu arbeiten und vermeiden sie Verantwortung.

Dabei geben Theorie X Manager ihren Mitarbeitern nicht die Gelegenheit, sich zu beweisen.

Die Führung übernehmen in den Organisationen des 20. Jahrhunderts Senior Manager und technische Experten. (McGregor 2005, INTRODUCTION TO THE ANNOTATED EDITION, Position 562)

Ein *Theorie Y* Manager hat die Annahme:

Mitarbeiter sind das Kapital der Firma und sollten wertgeschätzt und entwickelt werden.

Arbeiter sind motiviert, ihr Bestes zu geben und wollen Verantwortung übernehmen.

Das Bemerkenswerte an dieser Sicht ist, dass auch die Theorie Y es auch zulässt, im Einzelfall direktiv zu handeln. Die Einstellung des Managers ist ausschlaggebend.

Auch sind beide Grundannahmen dazu geeignet, selbsterfüllende

Prophezeiungen zu werden. Behandle ich Mitarbeiter als reine ausführende Organe, wird Eigeninitiative, intrinsische Motivation und Bereitschaft zur Verantwortungsübernahme verlernt. Gibt man Mitarbeitern die Möglichkeit zu wachsen, aus Fehlern zu lernen und Verantwortung zu übernehmen, so werden die meisten Leute dies auch tun.

Der Manager erzieht sich seine Mitarbeiter. Man könnte sagen „Jeder Vorgesetzte hat die Mitarbeiter, die er/sie verdient - zumindest langfristig." Dabei gehe ich davon, dass die Verteilung statistisch verteilt sein wird, aber der Mittelwert wird sich signifikant unterscheiden. Jetzt kann man dagegenhalten, dass ein Open Source Projekt nicht vom Verhältnis Führungskraft - Mitarbeiter ausgeht, sondern von der Kooperation auf Augenhöhe. Das stimmt nur zum Teil.

Wenn man eine Führungskraft als die Person definiert, an der keine Entscheidung vorbeigeht, so ist der Titel BDFL „Benevolent Diktator for Life", zu deutsch der „wohlwollende Diktator auf Lebenszeit" ein Hinweis darauf, dass es Unterschiede in der Entscheidungskompetenz gibt. Auch haben die meisten OS Projekte einen oder mehrere dedizierte Projektleiter.

Und genau das ist das Paradoxon bei Projekten Freier Software: Beteiligte (sogenannte Committer, also frei übersetzt Beitragende) machen in ihrer Freizeit, also als Hobby, das was andere als Arbeit machen.

Wo hört also die Arbeit auf und fängt der Spaß an? Nehmen wir folgendes konsistentes System von Annahmen, welches an den Taylorismus angelehnt ist:

- Arbeit ist nicht Spiel, Sport und Lernen
- Innerer Antrieb/Motivation gibt es nur bei Sport und Spiel
- (Mehr) Arbeit muss durch (mehr) monetäre Entlohnung motiviert werden
- Weil Arbeiter nicht arbeiten wollen, haben sie auch kein Interesse daran, wie die Arbeit verbessert werden kann

In diesem Denksystem kann es folgende Phänomene nicht geben:

- Hobbyarbeit oder Arbeitshobby = unbezahlte Teilnahme an OS Projekten
- Demotivation durch Belohnung = siehe 1.6.1 - Mehr Bezahlung - weniger Motivation

Also ist das Denksystem eher ungünstig oder die Umstände haben sich geändert.

1.5.3 *Drive*

In seinem Buch „Drive - The surprising Truth about What Motivates Us" (Pink 2010) untersucht Pink auch ein Beispiel aus dem „Open" Bereich, nämlich die

Erstellung einer Enzyklopädie.

Er lädt zu einem Gedankenexperiment ein. Die Fragestellung: Welches Enzyklopädie Projekt wird wohl erfolgreicher sein, eins oder zwei?

Projekt eins:

„Die erste Enzyklopädie kommt von Microsoft. Wie Sie wissen, ist Microsoft bereits ein großes und profitables Unternehmen. […]. Microsoft wird diese Enzyklopädie finanzieren. Es werden professionelle Autoren und Redakteure bezahlt werden, um von Hand Artikel über Tausenden von Themen zu erstellen. […]Manager werden das Projekt überwachen, um sicherzustellen, dass es budget- und termingerecht abgeschlossen wird. Dann wird Microsoft die Enzyklopädie auf CD-ROMs und später online zu verkaufen."

Projekt zwei:

„Die zweite Enzyklopädie kommt nicht von einer Firma. Sie wird von Zehntausenden von Menschen gebaut, die Artikel zum Spaß erstellen und bearbeiten. Diese Hobbyisten benötigen keinerlei besonderen Qualifikationen, um sich zu beteiligen. Und niemand wird einen Dollar oder einen Euro oder einen Yen dafür bekommen. Die Teilnehmer arbeiten manchmal zwanzig bis dreißig Stunden pro Woche kostenlos. Das Lexikon selbst, das online existieren wird, ist kostenfrei für jeden, der es nutzen möchte."

Mit klassischer Denkweise kommt man zu dem Ergebnis, dass bezahlte Profis, auf jeden Fall ein besseres Ergebnis bringen.

Doch es kam total anders - von Microsoft Encarta spricht niemand mehr, Wikipedia ist in aller Munde. Wie kommt das?

Der erste Denkfehler wäre es anzunehmen, dass sich nur Laien bei Wikipedia engagieren. Es sind vielmehr Spezialisten, die jeweils zu ihrem Spezialgebiet etwas schreiben.

Mit den vorher betrachteten Wirkmechanismen von open source Projekten fallen noch weitere Dinge auf: Wie Raymonds Basar gibt es bei Wikipedia sehr viele Tester der Qualität des Inhalts. Jeder Leser kann direkt Rückmeldung geben, was er für falsch hält.

Und auch hier werden die Diskussionen weltweit offen geführt. Wenn man auf einer beliebigen Wikipedia Seite auf die „Versionsgeschichte" klickt, so erhält man vollkommen transparent die Entstehung des Artikels selber sowie Diskussionen über den Inhalt.

Soweit ähnliche Wirkmechanismen. Es bleibt die Frage, was Beitragende zu offenen Projekten motiviert. Auf dem Weg, diese Frage zu klären, räumt Pink erst einmal mit dem Mythos auf, dass - ganz im Sinne des Theorie X Managers - Geld am Besten motiviert und die besten Arbeitsergebnisse bringt.

In Experimenten wird im Buch nachgewiesen, dass rein durch Geld von außen („extrinsisch") motivierte Menschen tatsächlich länger für eine Tätigkeit benötigen und dabei weniger kreativ sind.

Er entwickelt drei Hauptfaktoren, die Menschen bei nicht-langweilig-repititiven Aufgaben motivieren:

- Autonomie,
- Meisterschaft und
- Sinn.

Autonomie

Hier ist gemeint, dass zwar das Ergebnis vorgegeben ist, aber nicht der Weg, wie es erreicht wird. Das Verhalten wird dem Mitbearbeiter nicht aufgezwungen, sondern er ist frei (autonom), es auf seine Weise zu tun. Dies korrespondiert mit der fehlenden Reglementierung des Herstellungsprozesses und der Werkzeuge bei den OS Projekten.

Meisterschaft

Hier ist gemeint, dass das einzelne Individuum in seinem Bereich besser werden will, es also die Möglichkeit gibt zu lernen, bzw. sich zu entfalten.

Sinn

Hier geht es darum, dass auch bei vermeintlich kleinen Arbeiten an einem großem Ganzen dem Menschen vermittelt wird, wozu seine Arbeit sinnvoll ist.

Alle drei Faktoren sind bei OS Projekten gegeben. Es gibt keine oder nur wenig Reglementierung bei dem Herstellungsprozess, nur für das Ergebnis, also den Quellcode gibt es meistens Richtlinien. Diese sind von den meisten Committern als sinnvoll anerkannt.

Meisterschaft kann man durch den Austausch mit anderen Committern und dem erfolgreichen Lösen von Problemstellungen erringen. Durch den internationalen und vor Allem vielfältigen Austausch von Informationen kann man in direkter Kommunikation oder durch das Studieren der weltweit offenen Diskussionen zur Problemlösung vom Wissen und von der Herangehensweise an Problemlösungen profitieren.

Die Sinnfrage ist vorab geklärt. Nur wenn man als Beitragender Sinn in dem Projekt sieht, kommt man auch dazu. Sehe ich keinen Sinn in dem Projekt, werden ich auch nicht freiwillig beitragen.

Daher passt das Drive System von Annahmen besser, um die Mitarbeit an OS Projekten zu erklären.

Zu ähnlichen Ergebnissen bzgl. der intrinsischen Motivation kommt Trist (1990) in den „Grundsätzen der Arbeitsgestaltung“. In dem entwickelten sozio-technischen System werden die technischen äußeren Herausforderungen mit der inneren Motivation verbunden. Dazu die Vergleichstabelle im Überblick:

Extrinsisch	Intrinsisch
Angemessene und gerechte Bezahlung	Abwechslung und Herausforderung
Sicherheit des Arbeitsplatzes	Ständiges Lernen
Sozialleistungen	Eigenes Ermessen, Autonomie
Sicherheit	Anerkennung und Unterstützung
Gesundheit	Sinnvoller Beitrag in der Gesellschaft
Angemessener Arbeitsgang	Erstrebenswerte Zukunft
Arbeitsbedingungen	Die Arbeit selbst
Anstellungsbedingungen: sozial-ökonomisch	Tätigkeit selber: psycho-sozial

[1] Siehe Taylor, Frederick Winslow; 1903: Die Betriebsleitung - insbesondere der Werkstätten, Philadelphia. §22 […] Zweifellos ist der Durchschnittsmensch zu einem schlaffen und langsamen Tempo in allen Dingen geneigt, aus welchem er nur durch Besinnen auf sich selbst, durch Beispiele anderer oder durch äußeren Druck herausgebracht werden kann.

1.6 *Motivation*

1.6.1 *Mehr Bezahlung - weniger Motivation*

Wie auch in Herzberg (1977) ausgeführt, halten auch heutzutage viele Menschen an dem alten Glauben fest, dass Lohnerhöhungen Arbeitnehmer dazu motivieren, Arbeitsleistung zu steigern. Herzberg kommt zu dem Schluss, dass man Motivation nicht kaufen kann. Er illustriert das an dem Beispiel des Künstlers, der weiterhin Bilder malt, auch wenn er hungert.

Wiseman (2009) geht noch einen Schritt weiter: Er sagt, dass Belohnung hinderlich sein kann:

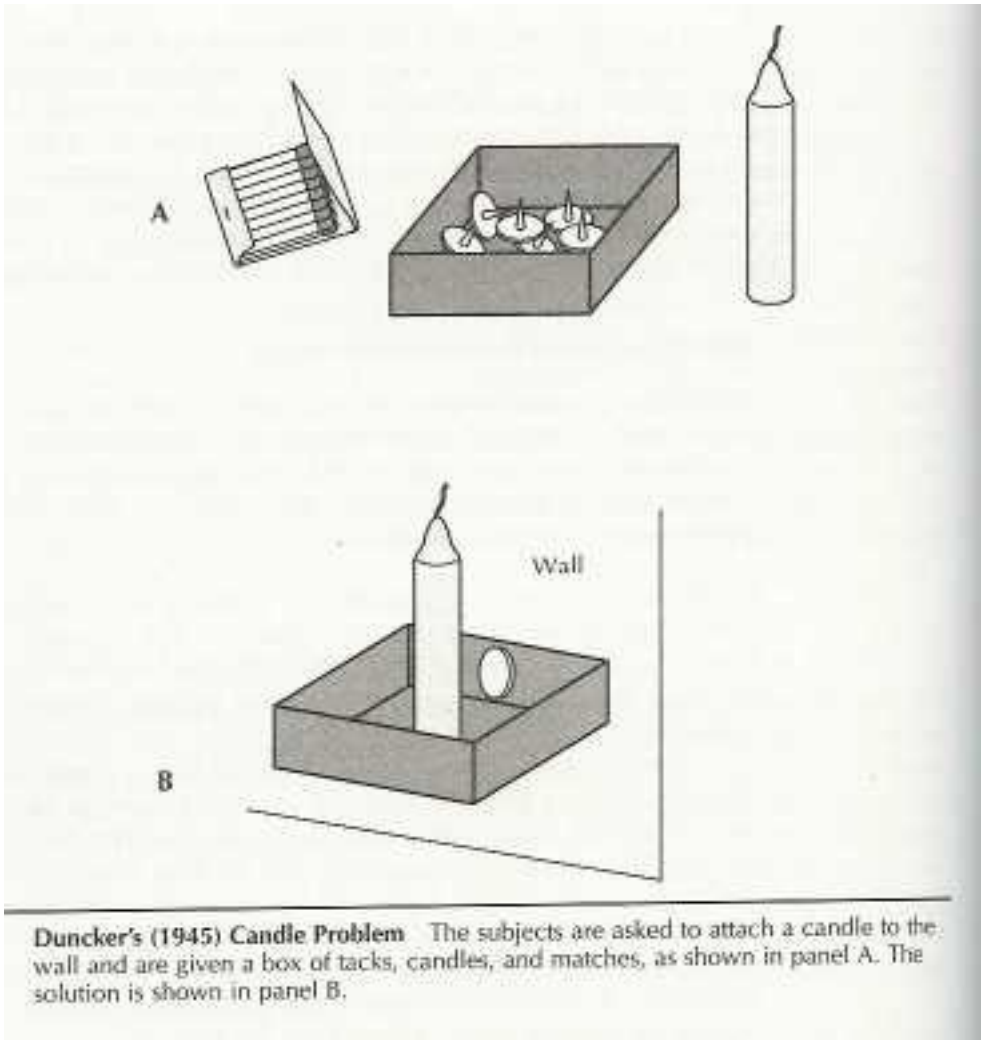
In einer berühmter Studie fragte der Stanforder Psychologe Mark Lepper und seine Kollegen zwei Gruppen von Kindern, dass sie Spaß daran haben sollen, ein paar Bilder zu malen. Bevor sie starten durften und mit den Stiften und dem Papier spielen durften, wurde der einen Gruppe gesagt, dass sie eine tolle Medallie „Guter Spieler“ bekommen würden. Der anderen Gruppe wurde keine Belohnung versprochen.

Ein paar Wochen später kamen die Forscher zurück, gaben Papier und Stifte aus und führten Messungen durch, wie lange die Kinder damit spielten. Erstaunlicherweise spielten die Kinder in der Gruppe, die das erste mal Medaillen erhalten hatten, signifikant kürzere Zeit als ihre Klassenkameraden.

Auch in Pink (2010) werden Studien[1] beschrieben, in denen Experimente zum Zusammenhang von zusätzlicher monetärer Belohnung und der benötigten Dauer zur Lösung komplexer Probleme durchgeführt wurden. Bei dem „Kerzen-Problem“, welches von Karl Duncker in den 1930ern entwickelt wurde, geht es darum, eine Wachskerze so an der Wand zu befestigen, dass sie nicht auf den Tisch tropft. Als Material stehen die Kerze, eine Schachtel mit Reißbrettstiften sowie Streichhölzer zur Verfügung.

Um auf die Lösung zu kommen, muss man die „funktionale Fixierung“ überwinden. Diese besteht darin, die Schachtel der Reißbrettstifte nicht mehr als Behälter für die Stifte zu sehen, sondern als Behälter für die Kerze. Befestigt man die Schachtel mit den Reißbrettstiften an der Wand, kann man die Kerze mit Wachs in der Schachtel befestigen und hat die Aufgabe gelöst.

Im Mittel benötigte die Gruppe, die mehr Geld für die schnelle Lösung bekommen hatte 3,5 mal länger für die Aufgabe, die von den meisten Leuten nach 5 bis10 Minuten gelöst wird.



Duncker's (1945) Candle Problem The subjects are asked to attach a candle to the wall and are given a box of tacks, candles, and matches, as shown in panel A. The solution is shown in panel B.

Bildquelle: Wikipedia

1.6.2 *Bezahlte Mitarbeit an Open Source Projekten*

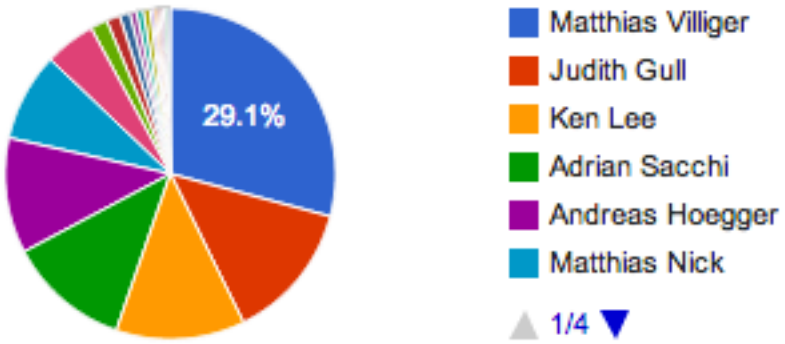
Gemeinhin wird angenommen, dass die Mitarbeit an OS Projekten unbezahlt sei. Hier ein paar Gegenbeispiele.

1.6.2.1 *Von kommerzieller Software zu Open Source - BSI AG und Eclipse Scout*

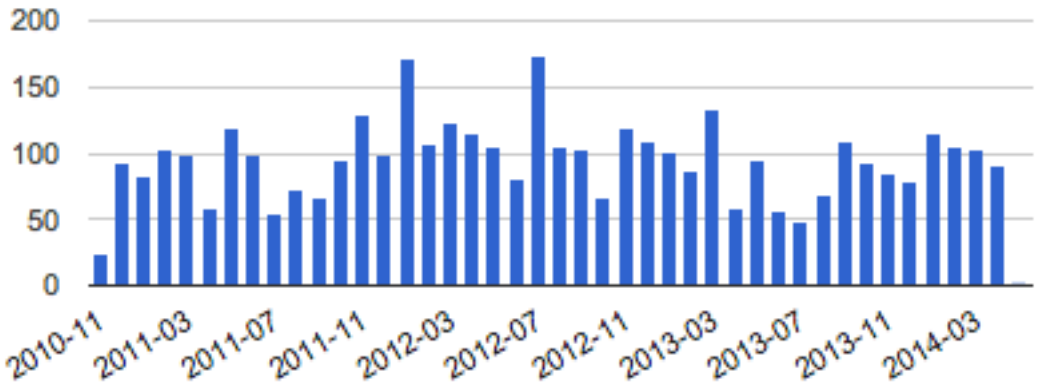
Die Schweizer Firma BSI[2] hat einen Bestandteil ihrer Basistechnologie als Open Source zur Verfügung gestellt: Das Eclipse Scout Framework[3]. Catherine Crowden von BSI nennt dazu auf der Website (siehe BSI (2010)) mehrere Beweggründe:

1. Investitionen sichern,
2. Herstellerunabhängigkeit,
3. keine Blackbox,
4. Mitentwickeln und
5. mehr Kontrolle für den Kunden, mehr Know How Aufbau beim Kunden.

2010 wurde Scout als Open Source verfügbar gemacht (siehe BSI (2010)). Auf der Projektwebsite[4] des Scout Projekts kann man sehen, welche Personen sich an dem Projekt in den letzten drei Monaten beteiligt haben. Dabei sind eine Vielzahl an Nicht-BSI Mitarbeiter vorhanden.



Prozentuale Individuelle „Commit“ Aktivität.



Anzahl an Beiträgen (Comits) über die Zeit.

Mitarbeiter bei BSI kümmern sich um die Projektleitung und werden so für die Mitarbeit an einem OS Projekt bezahlt.

1.6.2.2 *Von unbezahlter Mitarbeit zu angestellter Mitarbeit*

Linus Torvalds[5] arbeitete direkt nach dem Studium bei der Firma Transmeta, bei

der er auch während der Arbeitszeit weiter an Linux arbeitete. Später bekam er von RedHat, einem Unternehmen, welches auf Basis der von Torvalds entwickelten Technologie arbeitet, ein Aktienpaket, welches nach dem Börsengang von RedHat fast eine Million Dollar wert war.

Daneben nehmen sich die Spenden zu Anfang des Linux Projektes gering aus. Dazu ein Zitat[6] aus den Linux News von November 1992 unter dem Titel „Linus got some Money“ (Linus hat etwas Geld bekommen):

> *„November 6. Peter Anvin announced the final sum for the USA Linux fund collection, $785 and $750 after wire transfer charges.*
> *Linus thanks everybody, and reports that his computer has now been paid in full. He has also received other donations (both money and things), for which he is also grateful.“*

1.7 *Wie geht's weiter?*

Es wurden hier zuerst die Annahmen der Theorie X und Y betrachtet. Aber es gibt auch Projekt, in denen diese Annahmen irrelevant sind, da die Rollen Manager - Untergebener in der Form nicht existieren.

Im nächsten Abschnitt werden die verschiedenen historischen Stationen betrachtet, die eine politisch/moralische Einstellung von Freiheit genommen hat. Dies ermöglicht es, die heutigen Auswirkungen besser zu verstehen.

---

[1] “LSE: When Performance-Related Pay Backfires,” Financial, June 25, 2009.
[2] www.bsiag.com
[3] www.bsiag.com/de/blog/bsi/tags/scout.html
[4] http://projects.eclipse.org/projects/technology.scout/who
[5] http://de.wikipedia.org/wiki/Linus_Torvalds#Biografie
[6] http://www.ibiblio.org/pub/Linux/docs/ldpResearch/ldp-historic/LinuxNews.04A

2 Historie der freien Software

2.1 *1985 Free Software Foundation*

Am 4. Oktober 1985 gründete Richard Stallman die FSF, die „Free Software Foundation“, frei übersetzt die „Gesellschaft für freie Software“.

Was war der Hintergrund dieser Aktion? Stallmann arbeitete von 1971 bis 1984 am MIT am Labor für künstliche Intelligenz. Er entwickelte einen berühmten Unix[1] Texteditor, „Emacs“ genannt.

In den frühen Tagen der Computer waren die Hersteller mehr daran interessiert, Hardware zu verkaufen, so dass die Software weniger im Fokus stand. Wie Fogel (2009) schreibt, waren zum einen die Programmiersprachen jeweils nur speziell für eine Hardware gültig, zum anderen war es auch schwieriger, Software auszutauschen. Es gab noch kein Internet und die mobilen Speichermedien waren sehr begrenzt in ihrer Kapazität. Damit war der Kunde mit der Entscheidung für einen Hersteller stark an diesen gebunden.

Mir der Entwicklung der Programmiersprachen, die auf mehreren Rechner lauffähig waren, konnte das aufgebaute Wissen in einer Programmiersprache einfach für die Programmierungen auf anderen Rechner genutzt werden. Damit war die reine Hardware-Rechenleistung austauschbar und die mitgelieferte Software wurde als Unterscheidungsmerkmal wichtiger.

Damit hieß es für die Hersteller, die Software besser juristisch zu schützen und die Rechte (z.B. Copyright) strikter auf die Software anzuwenden. Dies war nicht in Stallmans Sinn. Wie Stallman (1998) schrieb:

„Wir nannten unsere Software nicht Freie Software, weil dieser Ausdruck noch nicht geprägt war, aber das ist das, was es war. Wann immer jemand von einer anderen Universität oder einer Firma ein Programm portieren und benutzen wollte, freute uns das und wir ließen sie gewähren. Wenn man jemanden ein unbekanntes interessantes Programm benutzen sah, konnte man immer den Quellcode bekommen, sodass man ihn lesen, verändern oder sogar Teile davon für neue Programme ausschlachten konnte.“

Dabei ging es ihm nicht darum, dass man den Quellcode der Programme sehen konnte, also das der „Source“ offen, also „Open“ war. Ihm ging es um die Freiheit, die Software selber verändern zu können. Deswegen verwendete

Stallmann auch den Begriff „Freie Software“ und nicht „OpenSource“. Auch der Kaufpreis spielt für ihn eine untergeordnete Rolle. Daher prägte sich der Spruch für freie Software ein:

„Frei wie in Rede, nicht wie in Bier“

Damit ist nicht das Reinheitsgebot von Bier gemeint, sondern Freibier als unbezahltes Produkt. Es ist also von untergeordneter Bedeutung, dass die Software kostenfrei ist. Wichtig ist, dass die darin enthaltene Information jedermann vollständig verfügbar ist. Wichtiger ist es, dass jeder ohne Beschränkungen auf den Quellcode und die Software zugreifen darf. Der freie Zugriff ist nicht nur erlaubt, sondern die Beschränkung des Zugriffs ist verboten. Daher haben OS Aktivisten und Softwareentwickler allgemein auch manchmal Schwierigkeiten mit der Beschränkungen auf Information. So werben die Verlage O'Reilly und Manning Publikation Co., die sich auf Bücher für Software spezialisiert haben auch damit, dass die elektronische Bücher ohne Zugangsbeschränkungen (DRM, Digital Rights Management) verkaufen, wobei Amazon, die eher den breiten Markt ansprechen mehr Bücher mit DRM verkaufen.

Im Januar 1984 kündigte Stallman seinen Job bei MIT, um ein großes Ziel anzugehen: Er wollte ein komplettes, freies Betriebssystem auf die Beine stellen. Dabei nannte er das Projekt „GNU“. Der Name selber ist ein Informatiker Witz, er ist ein rekursives Akronym. Das heißt er definiert sich selber. GNU bedeutet: GNU is not Unix.[2]

Das Fazit lautet: Wichtig sind die Werte Freiheit von Informationen, Zusammenarbeit und „Spaß“

Die Annahmen, die Stallmann über andere Comitter - die zu dem Projekt Beitragenden - stellt, kann man selber nur annehmen. Dadurch, dass aber die Transparenz und die Freiheit der Information das höchste Gut ist, kann man annehmen, dass Stallmann ein Theorie Y Manager ist. Wenn man annähme, dass andere Leute mit den Informationen nichts anfangen können, macht eine Transparenz der Informationen keinen Sinn.

Durchsucht man die Website (Stallman 2012) nach dem Wort Management, so taucht es dort nur in der Verwendung Verwaltung von Geräten oder als Referenz zu anderem Management auf. Hier ist anzunehmen, dass eher der kommerziell besetzte Begriff Management nicht zum Wortschatz gehört, als dass wirklich gemeint ist, dass es keine Steuerung gibt. Es gibt nur andere Arten von Steuerung. Bei der später betrachteten ASF gibt es ein klares Regelwerk zur Steuerung: „How the ASF works“. Stallman selber als „Manager“ der Free Software Foundation macht Management nicht zum Thema. Es geht primär um die Sache - freie Software, weniger um den Weg dahin.

[1] Ein Computer Betriebssystem, das früher nur auf Großrechnern lief und heute die Grundlage von Handys, z.B. Iphones und Google Handy ist.

[2] Es gibt auch weitere Tiere im GNU Projekt. Ein Programm heißt z.B. „Yacc“ - Yet another Compiler Compiler - und ein anderes Bison.

2.2 *1994 Linux 1.0*

Linus Torvalds hatte andere Beweggründe als Stallman. Wie er in dem gleichnamigem Buch schreibt, hat er das Projekt „Just for Fun“, also nur aus Spaß, begonnen.

Anders als bei Stallmann veränderte sich auch die Arbeitsorganisation während des Projekts „Linux“. Torvalds hatte das Projekt als kleines Programm begonnen. Nach ca. 6 Monaten fand er es ausgereift genug, um es ins Internet zu stellen. Schnell fanden sich mit vielen Unterstützern Team zusammen, welches unter der „Leitung“ von Torvalds Beiträge leisteten. Im Herbst 1992 lag die Zahl der Zugriffe auf das newsgroupforum comp.os.linux in der Größenordnung von 10.000 Leuten. Und das für ein Projekt, welches erst Anfang 1991 gestartet war!

Interessant auch hier, dass als Maßzahl für den Erfolg des Projektes die Anzahl der Kommunikationen genommen wird.

Zwei Themen waren es, die für die Zusammenarbeit und Nutzung wichtig waren. Zum einen die Rechte an dem Quellcode und die Art der Zusammenarbeit.

2.2.1 *CopyLeft*

Rechte am Quellcode - CopyLeft statt Copyright

Linus sagte zu den Rechten der Software

„Jedenfalls wollte ich Linux nicht verkaufen. Und ich wollte die Kontrolle darüber nicht verlieren. Das heißt, ich wollte auch nicht, dass andere es verkauften“

Als Inhaber der Urheberrechte legte er fest, dass man das Linux nicht verkaufen **darf** und dass man den Quellcode - auch von Änderungen jedem zugänglich machen **muss**. Das ist die genaue Umkehrung von „normalen“ Eigenschaften und Rechnet von Produkten. Es gibt Festpreise oder Mindestpreise - aber eine Einschränkung, dass man ein Produkt verschenken **muss** gab es bisher selten!

Linus stellte dann Linux unter das Copy Left der FSF/GNU, Das Lizenzsystem nennt sich „GPL“ - General Public License. Da es das Rechte auf freie Weitergabe schützt und nicht eindämmt - also genau andersherum läuft als ein „normales“ Copyright, wird es Copy Left genannt - wieder ein Informatikerwitz.

Linus Torvalds hat viele GNU Programme („Tools“) verwendet, um Linux ans Laufen zu bekommen. Aber er hatte nicht den moralisch/politischen Eifer eines Richard Stallmans. Linus sagte dazu:

„Aber im Gegensatz zu vielen hartgesottenen GPL-Freaks, die sich auf den Standpunkt stellen, jede Software- innovation solle unter der General Public License der Welt zugänglich sein, glaube ich, dass es den individuellen Erfindern selbst überlassen bleiben sollte, zu entscheiden, was sie mit ihrer Erfindung machen wollen.“

2.2.2 *Zusammenarbeit im Linux Projekt*

Zusammenarbeit im Linux Projekt

Torvalds sagt zur Zusammenarbeit in (Torvalds, 2002):

„Genau so wenig wie ich je geplant habe, dass Linux ein Leben außerhalb meines eigenen Computers führen soll, habe ich geplant, sein geistiger Kopf zu sein. Das ergab sich einfach so. Irgendwann fing eine Kerngruppe von fünf Entwicklern an, sich um die meisten Aktivitäten in den entscheidenden Entwicklungsbereichen zu kümmern. Es war nahe liegend, dass sie als Filter fungierten und die Verantwortung für die Pflege dieser Bereiche übernahmen.

Ich lernte ziemlich früh, dass du Leute am besten und effektivsten führst, indem du ihnen keine Vorgaben machst, sondern sie einfach ihre Vorstellungen umsetzen lässt. Die besten Führungspersönlichkeiten wissen auch, wann sie falsch liegen, und sind in der Lage, einen Rückzieher zu machen. Und die besten Führungspersönlichkeiten versetzen andere in die Lage, Entscheidungen für sie zu treffen."

Die Mit"arbeiter" im Linux Team waren absolut freiwillig dabei. Sie bekamen kein Geld dafür, nur Ruhm und Ehre und Spaß. Daher konnten Sie auch jederzeit wieder abspringen. Durch den Fachkräftemangel im IT-Bereich heutzutage sieht die Situation bei Firmen, die IT Mitarbeiter benötigen, fast ähnlich aus. Dennoch agieren die Firmen nicht nach der Situation.

Vergleichen wir der Bewerbungsprozess einer Arbeitstelle bei einer Firma als Informatiker mit der Mitarbeit im Linuxprojekt:

	Firma	Linux
Werbung	Die Firma wirbt z.B in einer Anzeige dafür, wie gut die Firma, die Arbeit und das Produkt ist.	Posting[1] im Diskussionsforum „Wer hat Lust mitzumachen".
Informationen über die Arbeitsweise der Firma/ Organisation	Informationen aus zweiter Hand: Text in der Stellenanzeige oder Imagebroschüre. Meist sind diese Texte durch Marketingabteilungen bis ins Detail ausformuliert und „auf Hochglanz poliert"	Informationen aus erster Hand: Interessenten sehen in den öffentlichen Diskussionen (z.B. In Mailforen)[2] den genauen Verlauf, wie Diskussionenen geführt werden und Entscheidungen getroffen werden.

Information über die Arbeitsweise	Handbücher und Regeln für die Art der Zusammenarbeit und offiziell dokumentierte oder inoffiziell gelebte Arbeitsabläufe. Das Ziel ist es meist diese Arbeitsabläufe zu vereinheitlichen	Minimale Regeln für die Zusammenarbeit. Regeln gibt es nur für das Produkt, also den Quellcode.
Entlohnung	Monetäre Entlohnung, eventuell Lob.	Offenes, im Internet weltweit sichbares Feedback zu Qualität der Arbeit. Nennung des Namens im Quellcode.

Dabei kann jeder Interessent auch zuerst zu Hause ausprobieren, ob ihm die Arbeit an dem Projekt gefällt. Dadurch dass tatsächlich das gesamte Produkt frei zugänglich ist, kann sich jeder, der es möchte, tatsächlich das gesamte Linux am eigenen Rechner zusammenbauen.

Das wäre übertragen auf Autos so, dass man alle Baupläne eines Motors inklusive Notizen der Entwickler des Motors legal aus dem Internet bekommt.

Damit könnte ich in den nächsten Baumarkt gehen und mir Motorteile kaufen - aber auch ausprobieren, ob nicht ein anderes Teil, z.B. ein Ventil nicht besser passt.

Wenn ich nun meine, das neue Ventil wäre besser, so schreibe ich an den Entwickler des Motors und sagt „Hey, ich hab da ein anderes Ventil ausprobiert, willst du das nicht einbauen?“[3]

Dieses Schreiben mit den Argumenten ist wiederum allen Menschen - nicht nur allen Mitarbeitern, sondern wirklich jedem, der es wissen will (und Internetzugriff hat) zugänglich. Im folgenden wird für diesen Grad der Offenheit der Begriff „weltweit offen“ verwendet.

Würde in diesem Fall der geistige Kopf des Projektes sagen: Ich will das Ventil nicht - und diese Entscheidung basiert nicht auf den Argumenten (die im Schreiben für alle frei zugänglich sind), sondern auf persönlichen Befindlichkeiten, dann wird es eine öffentliche Diskussion darum geben, ob die Entscheidung richtig war.

Beharrt nun der geistige Kopf auf seiner Entscheidung, von der mehrere Mitarbeiter glauben, dass sie falsch war, so können diese Mitarbeiter den kompletten Bauplan nehmen und eine neue eigene Produktreihe des Motors mit dem neuen Ventil aufbauen.

2.2.3 *Konkurrenz*

Diese neue Produktreihe steht nun in freier Konkurrenz zu dem ursprünglichem Produkt und Projekt. Dabei ist die Konkurrenzsituation mindestens auf drei Ebenen:

Konkurrenz beim Entstehungsprozess

Die Mitarbeit am Projekt muss den Teilnehmern gefallen, da sie sonst einfach nicht mehr teilnehmen oder sogar zum Konkurrenzprojekt gehen.

Konkurrenz beim Produkt selber

Die Mitarbeiter - und auch andere Experten in diesem Gebiet - sehen, welche Qualität das Produkt hat.

Übertragen auf den Motor im Beispiel sind wieder der Entstehungsprozess, alle Baupläne und auch der Motor, also das Produkt vollkommen transparent. Und es wird - offen für die Welt - darüber diskutiert, ob das Produkt gut ist.

Konkurrenz bei der Marktakzeptanz

Das tollste Produkt nützt nichts, wenn die Kunden oder besser gesagt die Benutzer es nicht kaufen oder besser gesagt verwenden.

Eines der bekanntesten Beispiele zum Thema Marktakzeptanz ist das der Videokassetten (Siehe „Formatkrieg“ aus Wikipedia im Anhang). In dem Wikipedia Artikel kann man nachlesen, dass Verfügbarkeit der Abspielgeräte, Marketing sowie die Reglementierung der Nutzung mehr zu dem Markterfolg beitrug als die Qualität der Produkte - hier Bildqualität und Spieldauer.

2.2.4 *Erfolgsfaktoren*

Drei weitere Aspekte, die auch für den Erfolg Open Source Projekte wichtig sind, möchte ich hier herausstellen:

Erfolgsfaktor „fehlende Reglementierung des Einsatzes“

In dem Beispiel der Videokassetten heißt es im Text

„Philips erlaubte nach einigen mündlichen Quellen keinen Vertrieb von Pornographie im VCR- oder Video-2000-Format.“

Ohne eine Betrachtung der moralischen Implikationen hat hier der Hersteller die Nutzung reglementiert. Leider wollten viele Kunden eben diese verbotene Nutzungsart.

Freie Software erlaubt die Nutzung unabhängig von der Einsatzart.

Wenn man den Einsatz von Linux z.B. unter http://en.wikipedia.org/wiki/List_of_Linux_adopters ansieht, so sind auch Organisationen wie das Militär

dabei Linux zu verwenden. Viele Menschen, wahrscheinlich auch Mitarbeiter am Linux Projekt haben Bedenken, für militärische Organisationen zu arbeiten. Aber auch diese Nutzung wurde für Linux nicht ausgeschlossen.

Der „Register“ schreibt Juni 2012:
„The US Navy has signed off on a $27,883,883 contract from military contractor Raytheon to install Linux ground control software for its fleet of vertical take-off and landing (VTOL) drones.“

Die anfängliche Abneigung gegenüber der politisch suspekten „freien Software“ ist also auch bei der US Navy fallen gelassen. Durch die fehlende Reglementierung des Einsatzes können auch Bereiche, an die der Urheber nicht gedacht hat, und auch Bereiche, mit denen der Urheber moralische, politische oder andere Schwierigkeiten haben könnte, die Softwareprodukte einsetzen.

Erfolgsfaktor „fehlende Reglementierung des Herstellungsprozesses“

Die Regeln, die sich die Entwicklerteams gaben, bezogen sich zumeist zuerst nur auf das Produkt, bzw. das Zwischenprodukt, den Quellcode. Welche Werkzeuge jeder einzelne für die Bearbeitung eingesetzt hat, ist offen. Auch hier gibt es z.B. bei den Quelltextbearbeitungswerkzeugen, also den Editoren einen offenen Wettbewerb. Auch der „Emacs“ des Richard Stallmann hatte seinen Gegenspieler im Programm „vi“.

Es ist auch freigestellt, wieviel Arbeit ein Teamarbeiter dazugibt. Es ist frei, wann das Teammitglied arbeitet. Dadurch, dass die Personen in verschiedenen Zeitzonen arbeiteten, wurde der zeitliche Prozess der Zusammenarbeit sowieso heterogener gestaltet.

Der größte Unterschied zu den Herstellungsprozessen der meisten kommerziellen Produkte, besteht in der Fokussierung auf den Zeitpunkt der Fertigstellung im Gegensatz zur Produktqualität. Ein Linux Kernel sowie die meisten Projekt der freien Software sind fertig, wenn alle Teammitglieder die Qualität und Funktionalität für sehr gut halten. Die Software ist fertig, wenn sie fertig ist. Das kann heute sein, aber auch in einem Monat.

Erfolgsfaktor „Offenheit bis zur Unhöflichkeit“

Torvalds sagte im Interview des Linux-Magazins 09/2011:

[…] Die andere große Schwierigkeit besteht darin, ein Projekt mit Tausenden Mitwirkenden zu organisieren – und mit Hunderten Firmen, die alle ganz unterschiedliche Zielsetzungen haben. Daher kam es in den vergangenen 20 Jahren immer wieder zu großen Meinungsverschiedenheiten zwischen Entwicklern. Wenn mir etwas an Linux schlaflose Nächte bereitet, dann sind es

Politik und persönliche Auseinandersetzungen.

Manchmal bin ich echt frustriert, dann schicke ich Flames an die Mailingliste und beschimpfe Leute. Zum Glück schaffen wir es meistens, unsere Probleme zu lösen, aber gelegentlich gibt es wirklich böses Blut oder ein Streit zieht sich über Monate hin.

Greg:Innerhalb von Firmen gibt es solche Streitereien aber auch – doch die internen Mails bekommt man ja nicht zu sehen.

Linus:Die Kernel-Mailingliste ist für ihren eher rauen Umgangston berüchtigt. Manche Leute schreckt das auch ab, weil sie sehr negative Antworten befürchten, wenn sie etwas einschicken. Beim Entwicklungsmodell des Kernels muss ich andererseits manche Dinge unmissverständlich klarstellen. Im Internet darf man kein Leisetreter sein, sonst kapiert keiner, was man möchte. Manchmal würde Höflichkeit einfach der Entwicklungsarbeit schaden.

Wenn ich vorsichtig schreibe "Das Patch braucht aber noch etwas Arbeit", richtet sich keiner danach. Deshalb schreibe ich: "Nein, zum Teufel! Diesen Code fasse ich nicht einmal mit der Kneifzange an!" Der manchmal aggressive Ton auf der Mailingliste bekommt der Entwicklungsarbeit einfach besser. Deshalb sind wir schonungslos offen und schreiben in unseren Mails Sachen wie: "Dein Code ist echt Mist – geh sterben!"

Aber selbst wenn ich mir einen Flame War mit jemandem geliefert habe, kann es sein, dass ich später meine Meinung ändere. Manchmal stellt sich einfach heraus, dass der andere doch recht hatte. Das ist zwar selten, kommt aber vor.[…]

Also ist hier Offenheit der Information wichtig für eines der erfolgreichsten Projekte der Freien Software.

---

[1] Textnachricht in einem Forum. Foren sind Diskussionsplattformen im Internet, die sich jeweils mit einem speziellen Thema beschäftigen.

[2] In Mailforen werden alle Mail zu einem Thema archiviert. Dabei sind alle Mails für alle Leser sichtbar.

[3] Tatsächlich gibt es bereits ein Projekt für ein Open Source Auto.

2.3 *1997 Die Kathedrale und der Basar*

1997 hat Eric Steven Raymond in dem vielbeachtetem Paper „Die Kathedrale und der Basar“ klassische Softwareentwicklung mit der Open Source Softwareentwicklung verglichen.

Dazu hat er die Vorgehensweise des Linux Projektes analysiert und die Faktoren, von denen er meinte, dass sie zum Erfolg maßgeblich beitragen, erfolgreich auf sein Projekt „fetchmail“ angewandt.

Dabei vergleicht er den geplanten Bau einer großen Software mit dem Bau einer Kathedrale. Bei dem Bau einer Kathedrale wird lange bis ins Detail geplant und dann nach dem Plan gebaut, bis die große Kathedrale fertig ist. Die ist dann aber auch heilig und perfekt. Das bedeutet, sie wird nicht geändert, da sie ja perfekt ist.

Die Entstehung eines Basars verläuft eher evolutionär: Ganz viele Leute laufen rum, bauen, schmeißen weg und bauen neu. Gar nichts ist heilig und alles kann passieren. Verschiedene Leute haben unterschiedliche Vorgehensweisen, nichts ist geregelt, es herrscht auf den ersten Blick Anarchie.

Ein wirklich erstaunlicher Aspekt dabei ist, dass dieser wilde Basarhaufen als Softwareprojekt schneller und stabiler neue Features entwickelte als jedes andere Projekt (Fogel 2009). Der oft scherzhaft zitierte Ansatz, dass 1000 Chinesen ein 1000 Manntage-Projekt in einem Tag erledigen, wurde wahr.

Die 19 Regeln, die Raymond dabei aufgestellt hatte, sind unten beschrieben. Dabei sagte Raymond, dass der Basar-Stil nicht für den Start eines Softwareprojekts geeignet ist, sondern eher für Fehlersuche und Erweiterung.

Eine wichtige Voraussetzung sind dabei eine große Anzahl an Usern, Entwicklern und Betatestern. Denn nur über die Masse der User kommt der sogenannte Delphi-Effekt zustande. Er besagt, dass die gemittelte Meinung der Masse von gleich kompetenten Beobachtern etwas bessere Voraussagen ergibt als die eines willkürlich herausgepickten Beobachters.

Auch hier stellt sich die Frage nach der Anwendbarkeit auf Projekte oder Produkte aus dem Nicht-Software-Bereich. Wenn ich die technischen gearteten Regeln Raymonds weglasse, bleiben zusammengefasst folgende Aussagen übrig:

I Sei bereit deine Meinung zu ändern, höre auf andere Meinungen (2,3,6,7,8,11,12).

II Mache nur Dinge, zu denen du motiviert bist (4,5,18).

III Sei bedingungslos transparent in Kommunikation und Information(7,10,19).

Auch hier bei I eine Abkehr von der managerzentrierten Theorie X, aber wiederum in II eine auf den ersten Blick egoistische Sichtweise. Genauer

betrachtet kann man Aussage II auch so verstehen, dass man dadurch, dass man nur an Dingen arbeitet, zu denen man motiviert ist, auch den Mit-Arbeitern[1] einen Gefallen tut.

Aussage III bezieht sich auch hier wieder auf die Transparenz der Information und des Wissens. Ein anderes Wort für Transparenz: Freiheit der Information!

2.3.1 *Richtlinien für gute Software*

Richtlinien für gute Open Source Software nach Raymond (1997)

1. Jede gute Software wird von einem Entwickler geschrieben, der ein persönliches Problem lösen will.
2. Gute Programmierer wissen, was sie schreiben müssen. Brillante wissen, was sie neuschreiben müssen (und was sie wiederverwenden können).
3. Plane, etwas wegzuwerfen; du wirst es sowieso tun. (Fred Brooks, „The Mythical Man-Month, Kapitel 112)
4. Mit der richtigen Einstellung finden dich interessante Probleme von alleine.
5. Wenn du das Interesse an einem Programm verlierst, ist es deine Pflicht, dieses einem kompetenten Nachfolger zu übergeben.
6. Wenn du deine Benutzer als Mitprogrammierer betrachtest, ist dies der einfachste Weg zu schneller Verbesserung und effizientem Debugging.
7. Veröffentliche früh. Veröffentliche häufig. Und höre deinen Kunden zu.
8. Mit einer hinreichend großen Gruppe von Betatestern und Mitentwicklern wird fast jedes Problem schnell erkannt und die Lösung von jemandem gefunden.
9. Schlaue Datenstrukturen und einfacher Code (im englischen Original: "dumb code") funktioniert viel besser als andersherum.
10. Wenn du deine Betatester wie deine wertvollste Ressource behandelst, werden sie dies auch werden.
11. Fast so gut wie eigene gute Ideen zu haben, ist es, gute Ideen von den Benutzern zu erkennen. Manchmal ist letzteres besser.
12. Meist entstehen die brillanten Lösungen aus der Erkenntnis, dass das Problem falsch verstanden wurde.
13. Perfektion (im Design) ist nicht erreicht, wenn man nichts mehr hinzufügen kann, sondern wenn nichts mehr entfernt werden kann.
14. Jedes Tool soll so funktionieren, wie erwartet. Aber ein wirklich gutes Tool führt zu Verwendungszwecken, an die du niemals gedacht hättest.
15. Wenn du Schnittstellencode schreibst, verhindere um jeden Preis, den Datenstrom zu verändern – und verwirf nur etwas, wenn dies der Empfänger verlangt.
16. Wenn deine Programmiersprache überhaupt nicht Turing-vollständig ist, kann syntaktischer Zucker dein Freund sein.
17. Ein Sicherheitssystem ist nur so sicher wie sein Geheimnis. Vermeide

Pseudogeheimnisse.

18. Um ein interessantes Problem zu lösen, suche eines.
19. Mit genügend guter Kommunikation, wie über das Internet, und Führung ohne Zwang sind viele Köpfe immer besser als einer.

---

[1] Mitarbeiter wird in konservativen Organisationen auch im Sinne von Untergebenen verwendet. Hier ist der ursprüngliche Sinn, also Kollegen auf der gleichen Hierarchiestufe gemeint

2.4 *Die Apache Software Foundation*

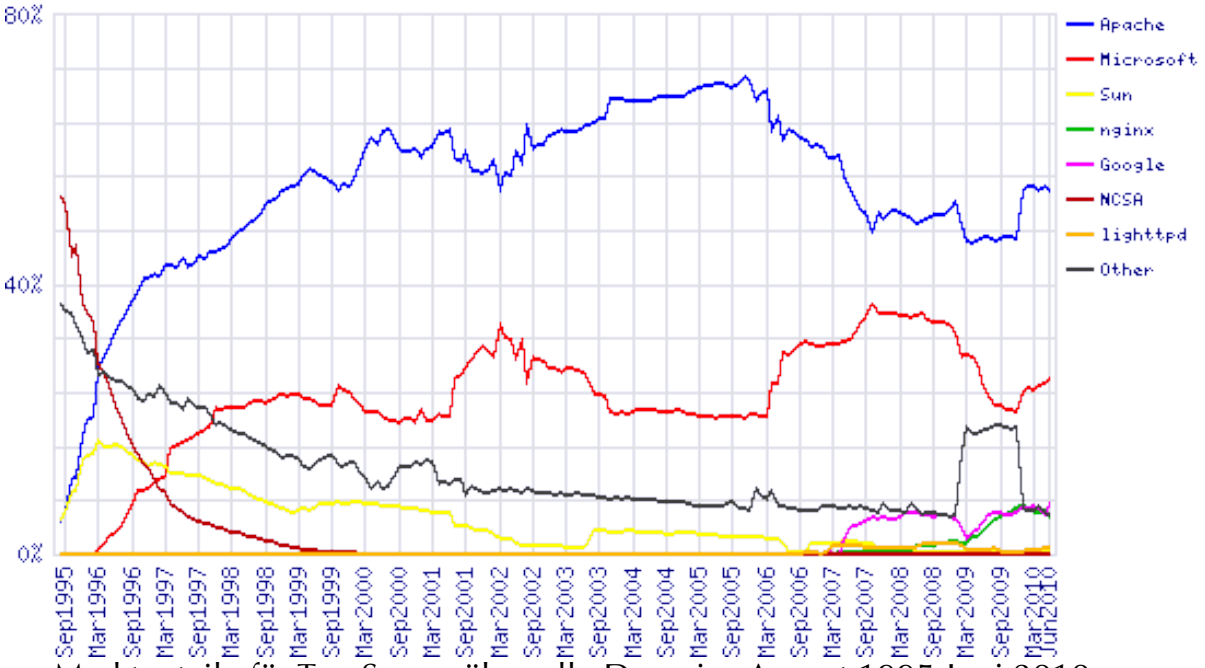
Im Februar 1995 war der populärste Web Server - also die Software, die Webseiten an Benutzer ausliefert, der „HTTP daemon“ von Rob McCool am National Center for Supercomputing Apllications der Universität von Ilinois. Da Rob McCool das Institut Mitte 1994 verlassen hatte, war die Weiterentwicklung des Webservers etwas eingeschlafen. Viele Verwender des Programms, also Webmaster - die Spezialisten, die eben diese Webserver betreuen, hatten kleine Änderungen und Erweiterungen für das Programm geschrieben, sogenannte „Patches“. Eine kleine, verteilte Gruppe dieser Webmaster verabredeten sich per E-Mail, da sie die Vielzahl dieser kleinen Patches zentral koordinieren wollten.

Ende Februar hatten sich acht Leute gefunden, sie gründeten die Apache Gruppe.

Der Name ist aus den ursprünglichen Patches entstanden, er bedeutet „ein gepatcheter Server“ - auf amerikanisch aPatche Server. Daraus wurde „Apache“ mit dem Federsymbol. Die sehr stabile und erfolgreiche Version 1.0 des Programmes wurde am 1. Dezember 1995 herausgegeben.

Kaum ein Jahr, nachdem die Gruppe sich zusammengefunden hatte, war der Apache Webserver die #1 im Internet. Das Open Source Programm hatte aller kommerziellen Webserver in der Beliebtheit, also der Anzahl der Installationen, hinter sich gelassen.

Das betraf - und betrifft noch heute - auch Produkte von großen Firmen wie Microsoft und Sun/Oracle. Das veranschaulicht auch die Grafik der Marktanteile für Webserver.



Marktanteile für Top Server über alle Domains August 1995-Juni 2010
Quelle: Netcraft (Market Share for Top Servers Across All Domains August 1995 - June 2010)

Die Netcraft Auswertung wurde im Juni 2010 über 206,956,723 Websites errechnet. In Zahlen: Zweihundertsieben Millionen Webseiten. Und davon gehen fast 60% an Apache! Ist dieser Erfolg einer Open Source Software ein Internetphänomen, oder lassen sich die Prinzipien und Arbeitsweisen auch auf andere Projekte und Unternehmungen anwenden? Um sich dieser Frage zu nähern, betrachte ich die Organisation und die Grundlagen etwas näher.

Die Apache Gruppe, aus der die heutige ASF - Apache Software Foundation hervorging, beherbergt heute über 100 Projekte. ASF sagt auf seiner Webseite, das sie nach dem Leistungsprinzip handeln - wer mehr Arbeit macht, hat auch mehr Rechte. Dabei dürfen alle interessierten Menschen oder auch Maschinen (weltweit offen) auf den Quellcode zugreifen.

Änderungen an dem Quellcode, also dem eigentlichem Produkt, werden auf der Mailliste vorgeschlagen und dann wird von aktiven Mitgliedern abgestimmt. Diese Maillingliste ist auch das primäre Kommunikationsmittel.

Geben die aktiven Mitglieder mindestens dreimal +1 (ein „Ja“ Votum) und kein −1 (ein „Nein“ Votum bzw. Ein Veto), dann darf die Änderung eingebaut werden.

Neue Mitglieder des „PMC“, des Project Management Committees werden hinzugenommen, wenn ein häufiger Beitragender[1] nominiert wird und einstimmig von den Abstimmungsberechtigten bestätigt wird.

Soweit klingt das alles wie aus der Satzung eines Kleingartenvereins, ist also irgendwie nichts neues. Das Prinzip, nachdem die Rechte gewährt werden ist jedoch neu - zumindestens in diesem Zusammenhang. Die ASF spricht hier von einer „Meritocracy“. Einfach übersetzt wäre das eine „Leistungs-Herrschafft“ -

gemeint ist jedoch mehr eine „Initiativ-Herschafft“. (Apache Software Foundation 2012 „How the ASF works“) Dazu ein Beispiel: Ein Entwickler darf Quellcode schreiben, wenn er vorab eine genügend große Menge an Dokumentation oder Code beigetragen hat. Er muss also zuerst Taten bringen, bevor er wirklich mitmachen darf.

Für die Entscheidungsfindung nennt die ASF eine „do-ocracy“ - eine Herrschaft der aktiv Beteiligten. Pragmatisch wird dazu auch gesagt „Es funktioniert in den meisten Fällen“. Es dürfen also nur Leute mitbestimmen, die beteiligt sind. Anders als in einer Demokratie dürfen aber nur Leute bestimmen, die *aktiv* beteiligt sind und damit die Auswirkungen zu spüren bekommen.

Ein bisschen steckt hinter diesen Namensgebungen meines Erachtens auch satirische Kritik an Wirtschafts- und Sozialwissenschaften.

Zwei Dinge sind in diesem Kontext noch zu erwähnen: Wie im Linux Projekt passiert die meisten Kommunikation öffentlich im Internet über Mailinglisten. Das ist eine Stufe Öffentlichkeit, die mehr ist als das was man normalerweise darunter versteht. Nicht nur, dass *jeder* (weltweit offen) mit einem Internetzugriff die Diskussion lesen kann, jeder kann sie auch noch Jahre später lesen. Das Internet kennt keinen Löschbutton.[2]

Dies zusammen mit der generellen Philosopie „respektvolle, ehrliche, technisch basierte Interaktion“ (respectful, honest, technical-based interaction) aus der Webseite Apache Software Foundation (2012): How the ASF works, bringt für die Diskussionen innerhalb der Apache Projekte offensichtlich positive Ergebnisse.

Die ASF sagt abschließend „In den ersten 10 Jahren des Betriebs hat die ASF ein sehr gutes Beispiel für eine offene Organisation gegeben, die die Balance zwischen Struktur und Flexibilität gefunden hat. Wir sind von 200 Beitragenden auf 3000 Beitragenden gewachsen. Wir haben mehrere Softwareprodukte, die Marktführer in ihrem Segment sind.“

Dabei sollte man folgenden Aspekt nicht außer Acht lassen: Wenn es um Erfolg geht, so findet man sehr viele Berichte und Bücher darüber „Mit dieser Methode haben Sie garantiert Erfolg - ich hatte es auch.“

Was ein wenig unterbesetzt ist, ist die Sparte „Mit der gleichen Methode hatte ich gar keinen Erfolg.“

Doch auch hier bietet die ASF Transparenz: Alle Projekte sollen mit den gleichen Grundlagen wie beschrieben arbeiten, sie können aber auch fehlschlagen. Viele Projekte sind der ASF in den Ruhestand (retired) gegangen. Zum Beispiel das Projekt „HiveMind“, welches ich selber verwendet habe. In der Abkündigung stand sinngemäß „die Mitglieder des HiveMind Projekt haben beschlossen, die Entwicklung einzustellen“.

Um mit der Aussagenlogik zu sprechen, die Transparenz ist für diese Art von Projekten notwendig, aber nicht hinreichend. Das meint, ohne Transparenz ist der Erfolg unwahrscheinlich, aber man braucht auch andere Bedingungen, z.B.

aktive Projektmitglieder und auch ein Problem, dass es mit der Software zu lösen gilt, also einen Sinn des Projekts.

2.5 *Fazit Historie*

Wir haben nun von 1985 bis 1999 verschiedene Stationen von Open Source Projekten betrachtet.

Bei allen Projekten nimmt die Transparenz der Kommunikation und die Motivation der Beteiligten die größte Bedeutung ein. Dies steht im Einklang mit den drei Motivationsfaktoren von Pink. Autonomie ergibt sich durch die Möglichkeit, eigenverantwortlich mitzuarbeiten. Meisterschaft ergibt sich durch die Zusammenarbeit mit anderen Experten und den Feedback, den die Experten und der Markt gibt.

Und der Sinn ergibt sich aus der ersten Richtlinie von Raymond: „Jede gute Software wird von einem Entwickler geschrieben, der ein persönliches Problem lösen will“.

Im Folgenden werden nun die Arbeitsformen von Open Source mit international verteilten Committern mit Arbeits- und Organisationsformen in kommerziellen Firmen verglichen.

---

[1] Im Original heißt es „Commiter“ oder Contributor. Im deutschen gibt es kein passendes Wort dafür.

[2] Dies kommt durch die vielen Kopien der Daten im Internet durch Suchmaschinen und andere Dienste. Wird der Originaltext z.B. in der Mailingliste gelöscht, so existieren meist mehrere Kopien des eigentlich gelöschten Textes noch an anderer Stelle.

3 Arbeitsformen

3.1 *Verteilte Arbeit*

Die Arbeitsformen für die OS Projekte werden in drei verschiedene Stufen betrachtet:

Stufe 1: Einzelarbeit

Bei R. Stallman steht primär Einzelarbeit im Fokus. Der Einzelne tauscht Informationen und Arbeitsergebnisse mit anderen Individuen aus. Dies liegt *nicht* am Unwillen, Informationen auszutauschen, sondern an Informationskanälen, die den sekundenschnellen, weltweiten Austausch noch nicht leisten können. Den Besitzer des OS-Projektes nennen wir Primärindividuum.

Stufe 2: Gruppenarbeit

Linus Torvalds fängt zuerst alleine an, sein Produkt (ist es wirklich ein Produkt?) vorzubereiten, dann wandelt sich die Arbeitsweise in ein internationales Team. Die Gruppe „Linux Entwickler“ hat durch das gemeinsame Ziel einen Zusammenhalt.

Stufe 3: Gruppen von Gruppen

Die ASF beheimatet verschiedene Gruppen aus Stufe 2.

Wer legt in den verschiedenen Stufen die Regeln fest? Klassisch/Theorie X betrachtet würde man sagen:

In Stufe 1 legt das Primärindividuum die Regeln fest.

In Stufe 2 legt das Primärindividuum die Regeln fest. Die Committer handeln nach den Regeln.

In Stufe 3 legt das Primärindividuum die Regeln fest. Die Primärindividuen der untergeordneten Stufe 2 Gruppen handeln nach den Regeln.

Die Wirklichkeit sieht anders aus:

In Stufe 1 stellt das Primärindividuum - im Folgenden PI genannt - die Regeln vor. Die Notwendigkeit, alleine zu Arbeiten besteht nicht mehr, da es schnelle und breite Informationskanäle gibt. Wenn jemand also alleine arbeitet, wäre

dies eine freie Entscheidung dazu. Das PI legt also die Regeln fest, arbeitet alleine und hält sich alleine an die Regeln.

In Stufe 2 setzt das PI Regeln fest. Wenn die Regeln den Committern nicht gefallen, wird hart diskutiert. Die Regeln werden angepasst oder die committer verlassen das Projekt. Wer Regeln festlegt, die keinem Comitter gefallen, fällt unfreiwillig zurück auf Stufe 1. Für die Bemessung der Qualität wird nicht nur das Produkt, sondern auch der Herstellungsprozess herangezogen. Dabei tritt der automatisierte Teil des Herstellungsprozesses in den Hintergrund und der Kommunikationsprozess der Gruppe wird wichtiger. Wie in Torvalds (2002) drängt sich hier die Anzahl der produktbezogenen Kommunikationen als Qualitätsmerkmal des Herstellungsprozesses auf.

In Stufe 3 legt eine Gruppe die Regeln fest. Diese werden öffentlich diskutiert und nach Abstimmung angepasst. Dabei nehmen die Diskussionsteilnehmer in den Gremien als Individuen teil, nicht als PIs. Das sieht die ASF als Stärke des Prozesses. Auch der Prozess der Regeländerung als Metaregel ist festgelegt. Auch hier wird abgestimmt. In Apache Software Foundation (2012) „How the ASF works“ werden als Philosophie 6 Meta Regeln als der „Apache Weg“ beschrieben:

1. Softwareentwicklung durch Zusammenarbeit
2. Standardlizenzen, die kommerziell nutzbar sind
3. Konsistente, qualitativ hohe Software
4. Respektvolle, ehrliche Interaktionen, die um die Technik gehen
5. Stringente Anwendung von Standards
6. Sicherheit als notwendige Eigenschaft

Auch hier wird der Fokus auf das Produkt gelegt. Nur zwei Regeln (1, 4) beziehen sich auf die Zusammenarbeit. Man kann davon ausgehen, dass diese Philosophie sich nur selten ändert. Durch die weltoffene elektronische Kommunikation und Interaktion kann ein Individuum, das sich überlegt mitzuarbeiten überprüfen, ob diese Philosophie auch gelebt wird. Das ist nicht per se „nett“, sondern konsequent transparent. Die Art der Zusammenarbeit - respektvoll - wurde ebenfalls festgeschrieben.

Eine der Ausgangsfragen dieser Betrachtung war die Frage: „Welche Erkenntnisse aus der Betrachtung der OS Projekte können eventuell auch für andere Projekte verwendet werden?“

Ich denke, dabei sollte man neben der absoluten Transparenz auch den evolutionären Charakter der OS Szene in Betracht ziehen. Jedes Projekt ist transparent in Bezug auf seinen Erfolg. Und dies wieder auf den drei Ebenen:

- Zusammenarbeit/Entstehungsweg
- Produktqualität und
- Marktakzeptanz

Unter der reinen Betrachtung der Produktqualität zeigt sich, dass Qualität von Produkten aus OS-Projekten meist besser ist als in kommerziellen Projekten. Um dies zu verstehen, machen wir einen kleinen Ausflug in die Theorie des klassischen Projektmanagements. Wie in Wikipedia (2013) Eintrag „Projektmanagement“ beschrieben, bewegt sich ein Projekt zwischen den Größen:

- Zeit: Projektdauer und Termine
- Kosten und
- Inhalt, Umfang und Qualität der Projektergebnisse.

Diese drei Größen stehen in einer Konkurrenz zueinander, beispielsweise in folgenden Situationen:

- Um den Termin zu halten, werden Überstunden geleistet und zusätzliches Personal beschäftigt; dies erhöht die Kosten.
- Um bei einem Projekt mit begrenztem Budget die Kosten zu halten, werden Leistungen gestrichen; dies senkt die Qualität der Ergebnisses.
- Um die Qualität des Projektergebnisses sicherzustellen, wird zusätzliche Zeit in Tests investiert und der Termin verschoben.

Soweit der Wikipedia Eintrag.[1]

Bei den meisten OS Projekten steht die Qualität im Vordergrund. Kosten sind fast egal, da die Beitragenden die Befriedigung nicht aus der (nicht vorhandenen) Bezahlung, sondern aus dem Ergebnis ziehen. Mischformen bei kommerziell unterstützen Projekten sind auch möglich. Die Kosten sind bei gleichbleibenden Hardwareeinsatz gering.

Die Zeit - bei vielen kommerziellen Projekten oft ein limitierender Faktor - gerät auch in den Hintergrund, da schneller fertig eben kein besseres Produkt ergibt.

Damit ist auch das erste Parkinsonsche Gesetz[2] für OS-Projekte außer Kraft gesetzt. Denn das „Gesetz“ besagt, „Arbeit dehnt sich in genau dem Maß aus, wie Zeit für ihre Erledigung zur Verfügung steht.“.

Da OS Projekten theoretisch unendlich Zeit zur Verfügung steht, würden sie nach Annahmen des Parkinsonschen Gesetzes nie fertig werden. Also bewegen sich OS Projekte außerhalb der Gültigkeit des „Gesetzes“.

---

[1] Wikipedia selber ist ein gutes Beispiel für OS-Projekte, bei denen der „Source“ auch die Information ist. Also das Paradies für Richard Stallman!

[2] Quelle Wikipedia:

Am bekanntesten ist das **Parkinsonsche Gesetz** zum Bürokratiewachstum, erstmals veröffentlicht 1955. Es lautet:

“Work expands so as to fill the time available for its completion.”

„Arbeit dehnt sich in genau dem Maß aus, wie Zeit für ihre Erledigung zur Verfügung steht.“
– und nicht in dem Maß, wie komplex sie tatsächlich ist.

3.2 *Hierarchie im Taylorschen Sinn*

In Stippler (2011) werden die unter Theorie X betrachteten Führungstheorien als „personenzentriert“ zusammengefasst. Dabei werden die Great-Man-Theorie, die Eigenschaftstheorie sowie die Skills Theory betrachtet.

Allen diesen Theorien gemein ist, dass die Fragestellung an sich bereits eine wichtige Vorannahme trifft, nämlich: „Der Erfolg einer Unternehmung hängt von dem Führer der Unternehmung ab“.

Beim „Great Man“ geht es um erfolgreiche Führungspersonen. Auch die Eigenschaftstheorie und die Skills Theory untersucht eben diese Personen.

Anfang dieses Jahrhunderts verloren diese Theorien an Bedeutung. Aufgegriffen wurde dies in dem Buch „The Tipping Point“ von Malcolm Gladwell (2002). Gladwell untersuchte die Faktoren für soziale Epidemien, wie wird ein Thema oder auch ein Produkt zum Erfolg, wann erreicht es den „Tipping Point“, ab dem eine vorher lineare, also gleichmäßige Entwicklung umkippt und sich stark beschleunigt.

Als eine von drei herausragenden Faktoren hatte er nicht die agierenden Personen, also Führungspersönlichkeiten, die es schaffen, durch ihre persönlichen Eigenschaften etwas zu bewegen, sondern die Umgebung, in der etwas geschieht, identifiziert.

Die anderen beiden Faktoren, oder Regeln, die er identifiziert hatte waren „das Gesetz der Wenigen“ und den „Anhänglichkeitsfaktor“. (Für Details verweise ich hier auf das Buch selber.)

Diesen Gedanken der Wichtigkeit der Umgebung für den Erfolg entwickelte er in dem Buch „Outliers“ (Überflieger) weiter (Gladwell, 2008). Er betrachtete unter anderem so erfolgreiche Personen wie die Beatles und Bill Gates.

Als notwendigen Faktor hatte er neben den durchaus vorhandenen Talenten auch die Möglichkeit, die Talente durch Übung zu entwickeln. Übrigens wurde hier als Faktor die 10.000-Stunden-Regel angewendet, also dass jemand 10.000 Stunden in einem Thema trainieren muss, um darin sehr gut zu werden.

Für diese Betrachtung ist die Erkenntnis wichtig, dass es nicht reicht, gut zu sein, um Erfolg zu haben, sondern, dass auch noch weitere Faktoren stimmen müssen.

Generell verschiebt sich die Sichtweise weg von dem Fokus auf eine einzelne Person, die durch ihre Fähigkeiten Erfolg ermöglicht, weg zu einer Gruppe von Personen, die Erfolg hat:

- Stallman war es wichtig, dass das Ergebnis seiner Arbeit frei verfügbar ist, und dass er frei auf andere Ergebnisse zugreifen kann. Also eine Art von verteilter Teamarbeit.
- Im Linux Projekt waren es schnell Teilverantwortliche, die sich um einzelne Bereiche des Projekts kümmern.
- Raymond hat in 7 von 12 Regeln betont, dass es wichtig sei, seine Meinung zu ändern.
- Und in der ASF darf der einzelne erst mitbestimmen, wenn er durch Mitarbeit seine Kompetenz bewiesen hat.

McGregor geht diesen Schritt nur halb. Er betrachtet die Vorannahmen, die ein Manager über seine Untergebenen trifft. Diese sind dann in der Theorie Y so, dass Arbeiten im Team erst ermöglicht wird. Er geht aber nicht soweit, die Führung selber zu dezentralisieren.

Ich denke, dass die implizit, getroffenen Vorannahmen hierbei eine zentrale Rolle spielen.

Arbeitsformen wie eine hierarchische Organisation, eine Matrixorganisation oder eine Projektorganisationen gehen von der Frage aus, wie man Führungspersonen strukturell verteilt, um effektiv arbeiten zu können.

Das Konzept „eine Person führt“ wird dabei nicht in Frage gestellt.

Das wie ich finde Interessante an den Open Source Projekten ist es, dass die Frage der Führungsstruktur zuerst *überhaupt nicht gestellt wird*[1], sondern sich entwickelt. Linux wurde zur Lösung eines Problems und aus Spaß, aus Eigenmotivation entwickelt. Als es reif war, wurde es der „Community“, also einer Gemeinschaft von an ähnlichen Themen arbeitenden Personen vorgestellt. Probleme der strukturellen Organisation wurden transparent in der Entwicklungsgemeinschaft geklärt. Und dabei wird selbstverständlich auf die Ergebnisse von anderen Projekten zurückgegriffen. Denn auch die Lösungen der Organisationsstruktur sind „frei“ im Sinne Stallmans.

Deswegen ist für die Betrachtung der Arbeitsformen meine ursprünglich gedachte Fragestellung „Wie ist die Arbeit in Open Source Projekten organisiert?“ auch einfach zu beantworten:

Sie entwickelt sich evolutionär!

Methoden, die erfolgreich angewendet werden, überleben und was nicht funktioniert, überlebt nicht.

Ein Ansatz wie in Theorie X wäre für diese OS-Projekte unpassend gewesen. Wenn man bedenkt, dass sich Personen aus ganz unterschiedlichen Kulturen weltweit zusammenfinden, um freiwillig an etwas zu arbeiten, dann kann man zu Recht annehmen, dass die Bereitschaft, sich dabei den Ideen einer einzelnen Person zu unterwerfen, relativ gering ist. Natürlich könnte es auch hier sein, dass ein gewisser Personenkult Leute dazu bringt, mitzumachen.

Die Transparenz aller Informationen gibt immer die Möglichkeit nicht nur das Produkt, an dem man eventuell mitarbeiten möchte, also die Software und den Quellcode, genau zu betrachten, sondern auch die Kultur der Kommunikation und die strukturelle Organisation. Übertragen auf den Prozess der Bewerbung eines potentiellen Mitarbeiters bei einer Firma hieße dies: Man darf vorab alle Emailkommunikationen lesen. Man darf vorab sehen, wie geführt wird und wie Mitarbeiter öffentlich dazu Stellung nehmen und auch wie Manager (um mal ein anderes Wort als Führungspersönlichkeit zu verwenden), antworten.

Und das ohne Mitarbeit einer Werbeagentur oder einer Unternehmenskommunikation, sondern die ungeschminkte Wahrheit, keine Interpretation von stattgefundenen mündlichen Kommunikationen, sondern die ursprüngliche Quelle, die EMail-Kommunikation.

Das ist ein höherer Grad von Transparenz als er in den meisten Organisationen vorzufinden ist!

---

[1] *Zumindest am Anfang nicht*

4 Fazit

4.1

Wie kommt es, dass augenscheinlich viele für wahr gehaltene Annahmen sich als falsch für den Erfolg von OS Projekten herausstellen? Und kann man diese Erkenntnisse auf kommerzielle Projekte übertragen?

(Gemäß dem Inhalt der Antwort erhebe ich keinen Anspruch auf Richtigkeit oder Wahrheit, sondern bin jederzeit dazu bereit, diese meine Meinung zu ändern, wenn es dafür einen Anlass gibt.)

Betrachten wir zur Beantwortung die drei nicht-technischen Wirkmechanismen des Basars:

1. Sei bereit deine Meinung zu ändern, höre auf andere Meinungen.
2. Mache nur Dinge, zu denen du motiviert bist.
3. Sei bedingungslos transparent in Kommunikation und Information.

Und kombinieren diese mit den Motivationsfaktoren nach Pink:

1. Autonomy
2. Meisterschaft und
3. Sinn

Sowie dem Kommunikationsstil eines Linus Torvals:

1. Offenheit bis zur Unhöflichkeit.

So stellen diese Thesen nicht nur wie in Theory y die Annahmen des Managers über andere, sondern auch über sich selbst, seine Position und das meiste Wissen, das er als richtig gelernt hat, in Frage. Um diese Haltung anzunehmen, muss man bereit sein, eine große Unsicherheit in Kauf zu nehmen, denn man kann sich nicht darauf zurückziehen, dass man doch alles „nach dem Buch“, also korrekt getan hat.

Bei dem evolutionärem Ansatz nimmt man das mögliche Scheitern der eigenen Anstrengungen und das der Organisation in Kauf.

Es wird viel Arbeit und Energie darauf verschwendet, es „richtig zu machen“ - also nach dem derzeit herrschendem Buch oder der Management Lehrmeinung zu arbeiten.

Mindestens genau so viel Energie wird verwandt, um das Selbstbild der Person und der Organisation aufrecht zu erhalten. Der Nebeneffekt ist es, dass Transparenz verloren geht.

Diese Meinung ist nicht neu und wurde auch nicht zum erstem Mal formuliert. Was in unserer Zeit relativ neu ist, ist die riesige Menge an Experimenten, die mit verschiedenen Arbeitsweisen zeigt, was auch funktionieren kann und was sogar besser funktioniert - nämlich eine Vielzahl an offenen Projekten, sei es Open Source Software Projekte oder Offene Wissensprojekte.

Wenn also OS-Projekte mit den beschriebenen Ansätzen erfolgreich sind, dann lohnt es sich diese Ansätze zu übernehmen oder zumindestens seine eigenen Ansätze kritisch zu prüfen.

5 Anhang

5.1 *Definition von freier Software*

Software gilt als frei, wenn:

- Sie die Freiheit haben, das Programm auszuführen, wie Sie möchten, für jeden Zweck;
- Sie die Freiheit haben, das Programm an Ihre Bedürfnisse anzupassen (um diese Freiheit in der Praxis umzusetzen, müssen Sie Zugang zum Quellcode haben, denn Programmänderungen ohne Quellcode sind außerordentlich schwierig);
- Sie die Freiheit haben, Kopien weiterzuverbreiten, entweder gratis oder gegen eine Gebühr;
- Sie die Freiheit haben, modifizierte Programmversionen zu verbreiten, damit die Gemeinschaft von Ihren Verbesserungen profitieren kann.

5.2 *Formatkrieg (Videorekorder)*

Formatkrieg (Videorekorder) - Nachdruck aus Wikipedia

> Dieser Artikel oder Abschnitt bedarf einer Überarbeitung. Näheres ist auf der Diskussionsseite angegeben. Hilf mit, ihn zu verbessern, und entferne anschließend diese Markierung.

Als **Formatkrieg** oder „Video-Krieg“ wird der Wettbewerb konkurrierender Videokassettensysteme der späten 1970er und frühen 1980er Jahre bezeichnet. Zu einem Zeitpunkt, als sich Heim-Videokassettenrekorder gerade zu einem industriellen Massenprodukt entwickelt hatten, existierten mehrere untereinander inkompatible Systeme, von denen sich dann etwa 1984 das VHS-Format weltweit durchsetzte.

Inhaltsverzeichnis [Verbergen]

Ursachen[Bearbeiten]

1971 starteten Grundig und Philips mit ihrem VCR das „Videozeitalter“ im Heimbereich. Die dazu benutzte Kassette hatte eine Spielzeit von rund einer Stunde, Zweikanalton und eine sogenannte *Color-under-Farbaufzeichnung*. Durch das Erscheinen eines japanischen Dreistundensystems im Jahr 1976 wurde die VCR-Kassette technisch überrundet, denn sie lief noch immer nur knapp über eine Stunde, während das japanische Konkurrenzprodukt VHS Filme in Spielfilmlänge ohne Unterbrechung aufzeichnen konnte. Die aufkommende Konkurrenzsituation zwang Philips und Grundig, ihr VCR-System kurzfristig weiterzuentwickeln. Er entstanden mehrere inkompatible Varianten, die bis zu fünf Stunden bei einer sehr akzeptablen Bildqualität liefen, und 1979 das Nachfolgersystem Video 2000 mit einer Spielzeit von zunächst zweimal vier

Stunden.

So konkurrierten nach nur kurzer Zeit folgende Videoformate: das europäische VCR-System von Philips und Grundig in drei untereinander inkompatiblen Varianten, die japanischen Formate VHS und ein weiteres, 1978 von der japanischen Firma Sony entwickeltes Zweidreiviertelstundensystem namens Betamax. Grundig und Philips wechselten ab 1979 völlig zu ihrer Neuentwicklung Video 2000 und gaben VCR im Consumer-Bereich auf. Alle Konzerne verfolgten dabei unterschiedliche Marketingkonzepte.

Marketing von JVC[Bearbeiten]

Alle Firmen weltweit, die unter ihrem eigenen Namen JVC-kompatible Videokassettenrekorder vertreiben wollten und weder Patente auf, noch Produktionskapazitäten für Videorekorder besaßen, legten mit dem Lieferanten JVC lediglich ihr Firmen-Layout und eventuelle Besonderheiten fest. Dann wurden vorerst alle Geräte, unabhängig vom Lizenznehmer, bei JVC in Japan produziert. Das System von JVC hieß VHS und startete in Europa Ende 1976.

Marketing von Sony[Bearbeiten]

Die Firma Sony bestand bei ihren Partnern darauf, dass diese eigene Produktionsstätten im jeweiligen Vertriebsland aufbauten, was naturgemäß lange Produktions-Anlaufzeiten schaffte. Danach lizenzierte man ihnen das Sony-Videosystem zum Nachbau. Dieses System Betamax wurde 1978 in Europa eingeführt.

Marketing von Grundig und Philips, die Situation in Westeuropa[Bearbeiten]

Philips N1500

Philips und Grundig waren die eigentlichen Pioniere der Heimvideosysteme gewesen. Ihr 1971 eingeführtes erstes Kassettensystem VCR bot eine Laufzeit von zunächst maximal 65 Minuten als Antwort auf das japanische Videoformat U-matic, das 1968 eine ähnliche Laufzeit pro Kassette zur Verfügung gestellt hatte, jedoch ausschließlich für semiprofessionelle Anwendungen gedacht war wie Schul- oder Industriefernsehen.

Im Heimvideobereich spielt eine lange Laufzeit eine bedeutendere Rolle als im Schul- oder TV-Bereich. Man wollte Spielfilme oder lange Unterhaltungssendungen am Stück speichern können. Erst 1977, kurz nach Erscheinen eines Dreistundenrekorders aus Japan, dem ersten VHS-Gerät der Victor Company of Japan (JVC), setzte man bei VCR dazu (neben einigen weiteren konstruktiven Änderungen) die Bandgeschwindigkeit herab und erreichte so eine Laufzeit von etwas mehr als zwei Stunden pro Kassette.

Da der japanische Konkurrent JVC aber im selben Jahr eine Videokassette mit

einer Laufzeit von vier Stunden ankündigte, entwickelte Grundig im Alleingang und ohne Philips das System nochmals weiter und schuf 1978 ein drittes, mit den vorausgegangenen Ein- und Zweistunden-VCR-Verfahren inkompatibles drittes Videosystem (Super Video Recordering, kurz: SVR). Es lief bis zu fünf Stunden. Gleichzeitig begann die Massenproduktion. Grundig errichtete im Nürnberger Stadtteil Langwasser für mehr als 50 Millionen DM eigens ein Videorekorderwerk für dieses neue Super-Longplay-VCR-Gerät.

Philips erreichte unter Verwendung eines dünneren Bandmaterials mit der älteren Zweistunden-Systemvariante von 1977 zeitgleich die Drei-Stunden-Marke. Die Grundig-Kassetten waren nun nicht mehr auf Philips-Geräten abspielbar. Käufer der Grundig-Geräte mit der mittellangen Laufzeit von 1977 konnten ihre Kassetten zwar auf den neuesten Geräten von Philips, allerdings nicht mehr auf den aktuellen Grundig-SVR-Rekordern mit der ganz langen Laufzeit abspielen.

Der technische Hintergrund dazu: Grundig hatte 1975 ein vollelektronisch gesteuertes, neuartiges Videolaufwerk entwickelt und setzte es bereits erfolgreich sowohl für die Ein- als auch die Zweistundenvartiante von VCR ein. Es war in der Herstellung äußerst kostspielig und geeignet auch für das Fünfstundensystem SVR. Das Philips-Laufwerk arbeitete noch immer – wie auch alle VHS-Geräte der damaligen Zeit – vollkommen mechanisch. Elektronisch geregelt war lediglich die Trommelrotation und der Bandvorschub. Die damit zu erzielende geringe Präzision reichte nicht aus, um damit ein SVR-Gerät im Hause Philips realisieren zu können.

Die Kunden des neuen Mediums reagierten verunsichert. Viele wandten sich von Grundig und Philips ab und kauften stattdessen die vorgenannten japanischen Produkte von JVC und Sony. Dazu rieten auch vielen Grundig- und Philips-Händler, die das Marketingkonzept um VCR Longplay und SVR nicht mehr verstanden.

Anstatt sich mit Grundig nun auf eine der drei Varianten zu einigen, bot man bei Philips und Grundig ab 1979 überraschend ein viertes, mit den bisherigen drei Systemvarianten erneut inkompatibles Format an, ein konzeptionell anderes, neues System. Es nannte sich Video 2000. Die Kunden blieben verunsichert, gerade in der wichtigen Periode des aufkommenden Massenmarktes für Videogeräte.

Akzeptanzprobleme von Video 2000[Bearbeiten]

Ein Video-2000–Rekorder

Die Folgen zeigten sich wenig später. Die überstürzte Markteinführung sowohl der Superlongplay-Version des VCR als auch kurz darauf des Video 2000 führte bei Grundig zu unausgereiften, unzuverlässig arbeitenden Geräten. Bei Philips gestaltete sich die Situation nur unwesentlich besser. Die Spieldauer des neuen Systems währte bis zu acht Stunden – nach vier Stunden wurde die Kassette umgedreht und konnte mit weiteren vier Stunden bespielt werden.

1981 war das VCR-System (abgesehen von einigen professionellen Geräten für Spezialanwendungen) vom Markt genommen, und es konkurrierten hauptsächlich VHS, Video 2000 und Betamax.

Ein wichtiger Punkt in dieser Zeit war die Verfügbarkeit von Miet- und Kaufkassetten. Videotheken richteten ihr Angebot auf das am meisten verbreitete System aus. Zu dieser Zeit war es bereits das VHS-Format. Und wie üblich bei neuen Medien war auch hier die Sex- und Pornofilmindustrie vorrangig vertreten. Philips erlaubte nach einigen mündlichen Quellen keinen Vertrieb von Pornographie im VCR- oder Video-2000-Format.

Bei einem Kaufpreis von mindestens DM 2200,- kam angesichts der Systemvielfalt bei den Verbrauchern spätestens 1979 große Unsicherheit auf. 1982 konkurrierten im Endverbrauchermarkt nur noch VHS, Betamax und Video 2000. Grundig hatte entgegen anderslautender Ankündigungen das SVR-System noch 1981 vom Markt genommen und brachte 1984 erste eigene VHS-Rekorder heraus. Der Marktanteil von VHS wuchs stetig.

1986 hatte sich VHS mit einem Marktanteil von 93 % durchgesetzt. Video 2000 kam noch auf 4 %, Betamax war auf 3 % gesunken.

Die Situation im Jahr 1989: Video-2000-Geräte werden nicht mehr produziert, der Herstellungsstopp liegt bereits drei Jahre zurück. Sony bietet noch Betamax-Geräte an, einen echten Käuferkreis gibt es nicht mehr. Die fabrikneuen Geräte waren wohl mehr für die großen Archive gedacht oder für außereuropäische Länder, in denen Betamax eine größere Bedeutung hatte erlangen können. Grundig und Philips verkauften ausschließlich VHS-Videorekorder.

Die Situation heute (2012): VHS ist noch immer das weltweit führende analoge Videosystem, man kann dafür auch heute noch neue Videokassetten und Geräte kaufen. Im Gegensatz zu Kassetten der Systeme VCR und Video 2000 sind auch Betamax-Kassetten zumindest noch auf Sonderbestellung erhältlich. Bei der Beschaffung von Videokassetten der Systeme VCR und Video 2000 ist man auf Restposten oder Gebrauchtware angewiesen. Meist weisen allerdings selbst die noch verpackten Altbestände große Lagerschäden auf und sind kaum noch zu verwenden.

Marktsituation 1980[Bearbeiten]

Marktanteile[Bearbeiten]

System	Marktanteil	Anbieter
VHS-Rekorder	53 %	JVC (Systementwickler), Akai, Blaupunkt, Graetz, Hitachi, Mitsubishi, Panasonic, Nordmende, SABA (zu Thomson), Sharpund Telefunken.
Beta-Rekorder	23 %	Sony (Systementwickler), Fisher, NEC, Sanyo, Toshiba und Wega.
Video-2000-Geräte	16 %	Grundig und Philips (Systementwickler), Bang & Olufsen, ITT, Ingelen, Körting, Loewe Opta, Metz und Siemens.
VCR-Rekorder, SVR-Rekorder	8 %	Grundig und Philips (Systementwickler), ITT und Siemens

Preise[Bearbeiten]

1980 kostete eine E240-VHS-Kassette umgerechnet etwa €27 bis €35, eine L195-Betamax-Kassette ca. €24 bis €27, eine VCR-VC60-Kassette (65 min/ 120 min / 240 min, je nach Systemvariante) rund €30 bis €38 und eine Video-2000-Kassette €31 bis €40.

Abgesehen davon waren Kassetten mit den genannten Spielzeiten nicht die gängigen Größen: Leicht verfügbar waren bei VHS die Länge E180 (180 min), bei Betamax L500 (120 min), bei VCR die VC30, die bei anderen Anbietern auch SVC-2 hieß (30/60/120 min) und die VCC 360 für das System Video 2000 mit zweimaligen 180 min.

Bei den Videorekordern hielten sich fast alle Verkäufer bis 1980 an die von den Herstellern empfohlenen Verkaufspreise. Diese lagen selten unter €1.500,-, meist eher weit darüber.

Beispiel aus dem Quelle-Winterkatalog 1978/79

- *Philips VCR Video-Kassetten-Rekorder N 1700 Long Play* mit einer VC 30: **DM 2.898,-**
- Einzelkassetten: VC 30 (30 bzw. 65 Minuten): **55 DM** — VC 60 (60 bzw. 130 Minuten): **75 DM** — VC 70 (70 bzw. 150 Minuten): **85 DM**
- *Akai VHS Video-Kassetten-Rekorder VS-9300*: **DM 2.989,-**
- Einzelkassetten: E-60: **39 DM** — E-120: **55 DM** — E-180: **65 DM**

Homevideo heute - nach einem weiteren Formatkrieg[Bearbeiten]

Ein moderner DVD-Rekorder

Die Einführung der DVD als Wiedergabe- und seit dem Jahr 2000 zunehmend

5.3 *Glossar*

Comitter - to commit - deutsch übergeben, einliefern, aber auch verpflichten. Personen, die zu einem Open Source Projekt beitragen, werden comitter genannt, da die Handlung „Quellcode an Projekt übergeben“ commit genannt wird. Es sind also Beitragende zum Projekt.

FOSS Free and Open Source Software - Freie und quelloffene Software.

FSF - Free Software Foundation. Gesellschaft für freie Software, gegründet durch Richard Stallman.

GPL - General Public License: wichtigste Rechteform für FOSS.

Hosten - Der Host ist der Gastgeber. Wenn ein Programm oder ein Projekt auf einem Rechner gehostet wird, so ist dieser Rechner der Gastgeber für das Projekt.

Open Source - Quelloffene Software. Das bedeutet, dass die Ursprungstexte, also der eigentlich Bauplan der Software offen verfügbar ist. Abkürzung: OS.

Quellcode - Der Text eines Programms. Meist wird der Text in ein auf einem Computer ausführbares Programm übersetzt.

Server - Computer, der Dienste bereitstellt. To Serve - bedienen.

5.4 *Quellen*

Apache Software Foundation (2012). *About*
Abgerufen 2.Mai 2014, von http://httpd.apache.org/ABOUT_APACHE.html

Apache Software Foundation (2012): How the ASF works
Abgerufen 2.Mai 2014, von http://www.apache.org/foundation/how-it-works.html

BSI (2010): Eclipse Scout offizielle Eclipse Ressource
Abgerufen 2.Mai 2014, von http://www.bsiag.com/de/medien/medienmitteilungen/medienmitteilungen-detail/article/eclipse-scout-als-offizielle-eclipse-ressource-freigegeben.html

Fogel, K. (2009). Producing Open Source Software. *Sebastopol: O'Reilly Media*

Gladwell, Malcolm (2002). The Tipping Point. *New York: Hachette Book Group*

Gladwell, Malcolm (2008). Outliers. *New York: Penguin Books*

Herzberg, Frederick; 1977; Motivation, Arbeitsmoral; in: Sonderdruck „Psychologie heute - Humanisierung des Arbeitslebens; Auszug.

Luczak, Holger at al. (1993). *Arbeitswissenschaft*. Berlin, Heidelberg: Springer-Verlag.

McGregor , D. (2005). *The Human Side of Enterprise, Annotated Edition*. [Kindle Edition]

Pink, Daniel (2010). *Drive - The Surprising Truth About What Motivates Us*. New York: Canongate Books.

Raymond, Eric S. (1997). *Die Kathedrale und der Basar.*

Thomson, I. , (2012), *US Navy buys Linux to guide drone fleet - Ground*

control to Uncle Linus.
Abgerufen 2. Mai 2014, von http://www.theregister.co.uk/2012/06/08/us_navy_linux_drones/

Trist, Eric (1990): Sozio-technische Systeme; Ursprünge und Konzepte; in: Organisationsentwicklung; 9(4)

Stalman, R. (1998), Übersetzung Stephan Knuth, (2003) ‚Über das Gnu Projekt"
Abgerufen 2. Mai 2014, von http://www.gnu.org/gnu/thegnuproject.html

Stalman, R. (2012). A Serious Bio
Abgerufen 2. Mai 2014, von http://stallman.org

Stippler, M. (2011). *Führung - Überblicke über Ansätze, Entwicklungen, Trends*. Gütersloh: Verlag Bertelsmann Stiftung

Sturm, A, Opterbeck, i., Gurt, J. (2011) . *Organisationspsychologie ; VS Verlag für Sozialwissenschaften | Springer Fachmedien Wiesbaden GmbH*

Torvalds, L. (2002). *Just for Fun: The Story of an Accidental Revolutionary*.

Wikipedia (2013):
Eintrag „Benevolent Dictator for Life"
Abgerufen 2. Mai 2014, *von „http://de.wikipedia.org/wiki/Benevolent_Dictator_for_Life"*
Eintrag „Parkinsonsche Gesetze"
Abgerufen 2. Mai 2014, *von „http://de.wikipedia.org/wiki/Parkinsonsche_Gesetze"*
Eintrag „Projektmanagement"
Abgerufen 2. Mai 2014, *von: http://de.wikipedia.org/wiki/Projektmanagement*
Eintrag „Projekt"
Abgerufen 2. Mai 2014, *von http://de.wikipedia.org/wiki/Projekt*

Wiseman, Richard (2009). *59 Seconds Think a little, change a lot*. New York :Alfred A. Knopf