Der Nutzen des Konzeptes Open Source bei der Softwareentwicklung am Beispiel des Linux-Kernels

Autor: Josua Brandhofer

In: Informatik (GK1)

Betreuungs-/Kurslehrer: Herr Berger

Name der Schule: Siegtal-Gymnasium Eitorf

Abgabedatum: 20.02.2015

Inhaltsverzeichnis

1 Einleitung

Computer sind aus dem heutigen Leben kaum noch wegzudenken. Sie werden zur Freizeitgestaltung oder bei der Arbeit genutzt. In den Industrieländer hat beinahe jeder Haushalt einen Computer. Hinter dem Computer steckt nicht nur die bloße Hardware. Noch viel wichtiger dabei ist die Software. Hinter der Softwareentwicklung oder den IT-Dienstleistungen steckt ein großer Markt, welcher immer wichtiger wird. Solche Software kann ein Spiel, ein Programm zur Steuerung komplexer Anlagen oder ein ganzes Betriebssystem sein. Als bekanntestes Betriebssystem gilt dabei das proprietäre Betriebssystem „Windows“ von Microsoft. Dieses wird kommerziell vertrieben und wird auf den meisten Privat Computern vorinstalliert. Daneben gibt es aber auch die sogenannte Open Source Software. Prominente Beispiele dafür sind z. B. der Internet Browser „Mozilla Firefox“ oder das Betriebssystem „Linux” mit all seinen verschiedenen Distributionen. Diese können von jedem, anders als bei „Windows“, unentgeltlich genutzt und weiterentwickelt werden. Meistens wird „Linux“ aber eher versteckt genutzt. So nutzt man eigentlich täglich Linux wenn man „Google.com“ benutzt, da die Server von Google mit einem Linux-Betriebssystem laufen. Auch bei Supercomputern wird Linux sehr häufig genutzt. Aktuell werden 94,6% der Supercomputer zumindest teilweise mit Linux betrieben[1]. Bei diesem häufigen Vorkommen kam auch Herr Prof. Dr. Schwalbe zu dem Schluss: „Somit wird deutlich, dass auch Open Source Software für viele Bereiche des Lebens bedeutend ist und aus der heutigen, durch Informationstechnologien geprägten Welt, kaum noch heraus zu denken ist.“ (Schwalbe, 1. Einleitung). Bei so einem großen Erfolg von Open Source Software stellt sich die Frage: Inwiefern hilft das Konzept von Open Source Software ein so gutes Produkt[2] zu erstellen, welches häufig kostenlos ist? Inwiefern ist dieses Konzept vom Ergebnis her besser als proprietäre Software? In dieser Facharbeit möchte ich mich mit dieser Frage auseinandersetzen und dabei mehrere Texte untersuchen. Dabei werde ich mich am Beispiel des Linux-Kernels orientieren. Zu Anfang werde ich erstmal eine kurze Definition von Open Source Software geben und sie dabei von anderen prominenten Softwaretypen abgrenzen. Zusätzlich gebe ich eine kurze Darstellung des Linux-Kernels. Darauf werde ich im 4. Kapitel die Geschichte von Open Source Software erörtern, wobei ich mich auf die des Linux-Kernels konzentriere. Im nachfolgenden Kapitel werde ich den Nutzen des Open Source Konzepts für verschiedene Beteiligte untersuchen. Dies wären das Projekt an sich, die Verbraucher, für die die Software gemacht wurde, die teilnehmenden Entwickler und schlussendlich die Firmen, die die Open Source Software nutzen oder mitentwickeln. Im letzten Kapitel werde ich mich mit Kritik

[1] vgl.: Wikipedia; Supercomputer; 2015; Abschnitt: Betriebssysteme

[2] So gut, dass Microsoft es als große Bedrohung ansieht(siehe auch Halloween-Dokumente)

an Open Source Software bzw. deren Problemen auseinandersetzen.

2 Definition Open Source Software und Abgrenzung

Open Source Software ist, wenn man es wörtlich übersetzt, Software, deren Quelltext offengelegt ist. Man sollte dies aber nicht mit gemeinfreier Software, welche nicht lizenziert wird und allen kostenlos zugänglich ist, verwechseln. Denn Open Source Software wird sehr wohl lizenziert. Dabei gelten laut der Open Source Initiative (im folgenden OST) für Lizenzierung im Sinne von Open Source folgende 3 Grundmerkmale (laut Wikipedia, Open Source, Abschnitt: Definition der OSI): 1. „Die Software (d. h. der Quelltext) liegt in einer für den Menschen lesbaren und verständlichen Form vor.“, 2. „Die Software darf beliebig kopiert, verbreitet und genutzt werden.“ und 3. „Die Software darf verändert und in der veränderten Form weitergegeben werden.“[3]. Also gibt Open Source Software seinen Nutzern weitaus mehr Freiheiten bei der Softwarenutzung als proprietäre Software, welche ihren Quelltext meistens geheimhällt. Diese weitergehenden Rechte bedeuten aber nicht, dass die Software unbedingt kostenlos angeboten wird. Die mögliche Missinterpretation des Wortes frei in der zur Open Source Software weitgehend identischen freien Software hat primär zur Entstehung der OSI beigetragen. Daher stammt auch die Aussage der Free Software Foundation (FSF) über die Bedeutung des Wortes „frei“: „free speech, not free beer“(Redefreiheit, nicht Freibier). Nichtsdestotrotz wird sehr viele Open Source Software kostenlos angeboten[4]. Die meisten Open Source Projekte werden unter der von der FSF erstellten GNU General Public License veröffentlicht.
Open Source muss man zwangsläufig zu der sehr ähnlichen freien Software abgrenzen. Diese sind zwar sehr ähnlich, nur garantiert freie Software mehr Freiheiten für den Nutzer als Open Source Software[5]. Der größte Unterschied besteht aber in der Bedeutung der Begriffe und in den Werten, den sie vermitteln: Open Source zeigt die freie Verfügbarkeit des Quelltextes. Auch stellt die OSI primär den praktischen Nutzen dieser Entwicklungsmethode heraus und zeigt eher technische und wirtschaftliche Vorteile. Die freie Software nach der FSF vertritt eher die Rechte der Nutzer an der Software und zeigt auch sozial ethische Aspekte auf. So geht es ihr bei der Ablehnung proprietärer Software nicht bloß darum, wie gut sie trotz nicht öf-

[3]Bei mancher Software wird ergänzend zum 3. Merkmal noch weiterführend das Copyleft-Prinzip angewendet, um auch weiterhin die Rechte des Nutzers zu wahren. Dabei wird bewusst auf das Urheberrecht verzichtet.

[4]Das Unternehmen ID Software geht dabei einen interessanten Weg. Sie machen den Quelltext ihrer Spiele öffentlich zugänglich, z. B. auf GitHub, zugänglich und verlangen nur Geld für die graphischen und musikalischen Inhalte.

[5]Was nicht bedeutet, dass Open Source Software diese nicht bieten kann.

fentlichen Quelltextes auch sein mag, sondern behauptet, „dass proprietäre Software allein schon aus moralischen Gründen abzulehnen sei[6]“. Insgesamt sind sich Open Source und freie Software sehr ähnlich. Eigentlich ist es nur ein Konflikt aufgrund der Bezeichnung. Deshalb werden auch Begriffe wie „Free/Libre and Open Source Software“ (FLOSS) genutzt.

3 Was ist der Linux-Kernel?

„Linux ist ein Betriebssystem-Kernel. [...] Der Kernel wird in einer Vielzahl von Betriebssystemen genutzt, die oft selbst als Linux bezeichnet werden.“ (Wikipedia, Linux(Kernel)). Einer der Hauptentwickler des Linux-Kernels ist Linus Torvalds. Als Kernel ist der Linux-Kernel eines der wichtigsten Teile des Betriebssystem. Er sorgt dafür, dass eine einheitliche Schnittstelle für die Software zur Verfügung steht, welche Hardware unabhängig ist. Der in weitgehend in C geschriebene Kernel ist ein strikt monolithischer Kernel[7]. Des weiteren bietet der Kernel 4 verschiedene Schnittstellen: die in jedem Fall stabile externe Programmierschnittstelle, die nicht stabilitätsgarantierte interne Programmierschnittstelle, die „Linux Standard Base[, welche eine Binärschnittstelle ist,] soll es ermöglichen kommerzielle Programme unverändert zwischen Linux Betriebssystemen zu portieren“ (Wikipedia, Linux(Kernel)) und schlussendlich die nicht stabile interne Binärschnittstelle. Obwohl es eigentlich nicht so geplant war, hat sich der Linux-Kernel weitestgehend in Richtung eines portierbaren Kernels entwickelt und gehört zu einem der am häufigsten portierten Systemen. Der Kernel ist seit 1992 unter der freien GPL lizenziert. Als Linux bezeichnet man Betriebssysteme, die auf dem Linux-Kernel und im Wesentlichen auf GNU-Software basieren. Auf Computern kommt Linux in Form einer Distribution vor. „Eine Distribution fasst den Linux-Kernel mit verschiedener Software zu einem Betriebssystem zusammen, das für die Endnutzung geeignet ist.“ (Wikipedia, Linux). Die hohe Anzahl an Linux-Distributionen bietet dem Endnutzer die Möglichkeit, sein auf ihn persönlich zugeschnittenes Linux zu erhalten. Das auffälligste Merkmal an Linux ist seine hohe Sicherheit vor Viren und dergleichen. Dies wird durch eine strenge Unterteilung der Zugriffsrechte erreicht, so dass die Verunreinigung des gesamten Systems schwer ist. Auch die schnelle und häufige Aktualisierung hilft, Sicherheitslücken zu schließen.

[6] Wikipedia, Freie Software, Abschnitt: Open Source

[7] D. h. werden der Quellcode und alle Treiber in das Kernelimage kompiliert. Das Kernelimage ist der ausführbare Kernel.

4 Geschichte von Open Source und die des Linux-Kernels

4.1 Vorgeschichte

Am Anfang der Computerindustrie wurde Software noch nicht als eigenständiges Wirtschaftsgut behandelt. Die Software wurde als Teil des Rechners behandelt und individuell auf den Kunden angepasst. Der Quelltext wurde oft mit der Software mitgeliefert. Es hatte also Eigenschaften von heutiger Open Source Software. Ca. zwischen 1960 und 1970 entwickelte sich an Universitäten und Forschungseinrichtungen eine so genannte „Hacker-Kultur“. Diesen leidenschaftlichen Programmierern ermöglichte die freie Verfügbarkeit des Quelltextes das Austauschen ihrer Programme und die gemeinsame Weiterentwicklung. Dies hatte für die Computerhersteller den Vorteil, dass sie „viele Vorschläge für Verbesserungen und Fehlerkorrekturen“[8] zurück bekamen.
In den 1970er Jahren began die Kommerzialisierung der Software. Wegbereitend dabei war das Unternehmen IBM, welches bis 1970 seine Software kostenlos veröffentlichte. „Am 23. Juni 1969 kündigte IBM neue Regeln für die Nutzung und Wartung seiner Software, getrennt von den Hardware-Nutzungsbedingungen an. Für Software wurde urheberrechtlicher Schutz in Verbindung mit Lizenzverträgen eingeführt.“[8] Andere Firmen zogen nach und so entstand der Wirtschaftsmarkt für die Software und damit auch Softwareunternehmen, die zwar nun auch hardwareunabhängige Software herstellten, aber sie „verfolgten mit der Entwicklung von Software kommerzielle Ziele, die dazu führten, dass der Großteil der Software proprietär war und nur in kompilierter Form veröffentlicht wurde“[9]. Dies machte die Anpassung an eigenen Bedürfnisse und die private Weiterentwicklung unmöglich. Des Weiteren schränkten Softwarelizenzen auch noch Nutzungszweck und Weitergabe ein.
Ein weiteres Beispiel ist die Entwicklung Betriebssystem Unix. Es ist deshalb wichtig, weil die Geschichte von Unix stark mit der Geschichte von Linux zusammenhängt. Unix wurde von der Telefongesellschaft AT&T entwickelt. Es durfte bis 1984 Unix nicht gewerblich vertreiben. Dies war für diverse Universitäten von Vorteil, da sie Unix so benutzen und für ihre Zwecke anpassen konnten. Daher bildeten sich in den 1970er Jahren „Abspaltungen vom ursprünglichen AT&T Unix heraus“[9]. In den 1980er Jahren wurde es AT&T möglich bzw. beschlossen sie, AT&T Unix als proprietäre Software zu vermarkten. „Infolgedessen durfte nun auch der AT&T-Quellcode nicht mehr öffentlich zugänglich gemacht werden.“[10] Dies führte dazu, dass sämtliche Abspaltungen vom ursprünglichen Unix (allen voran BSD Unix) ihren Quelltext von

[8] Wikipedia, Freie Software, Abschnitt: Entwicklungen im Vorfeld
[9] Schwalbe, 2.3.2
[10] Wikipedia, Geschichte von Linux, Abschnitt: Entwicklungen im Vorfeld

AT&T Unix Quelltext bereinigen mussten um Lizenzgebühren zu entgehen, was die Softwareentwicklung sehr behindert.

4.2 Freie Software und Open Source Software

Als „Vater der Freien Software“ gilt Richard Stallman. An seinem Arbeitsplatz im Labor für künstliche Intelligenz des Massachusetts Institute of Technologie wurde in den 1970er Jahren immer mehr proprietäre Software eingeführt. Dabei störte ihn unter anderem die fehlende Möglichkeit auf den Quelltext zuzugreifen und so Fehler zu beheben. Auch die so entstehende „Monopolstellung proprietärer Anbieter“ stoß ihn ab. So versuchte er zuerst, „durch das Programmieren alternativer Software“ dem Entgegenzuwirken. Dabei wollte er einen „freien und ungehinderten Austausch von Software“ erreichen (Wikipedia, Freie Software, Abschnitt: Entwicklungen im Vorfeld). Da die Kommerzialisierung der Software weiter voranschritt, wollte er als Basis für „ein freies Software Universum“[11] ein freies Betriebssystem entwickeln. So kündigte er im September 1983 das GNU-Projekt an und startete es schließlich im Januar des darauf folgenden Jahres. Während dieser Zeit entwickelte er auch das Copyleft-Prinzip. Um eine Basis zu haben gründete er 1985 die Free Software Foundation und 1989 veröffentlichte er die GNU General Public License, welche „die heute am stärksten verbreitete Lizenz für Freie Software“ ist (Wikipedia, Freie Software, Abschnitt: Die Entstehung Freier Software). Bis zum Jahr 1991 stellte das GNU-Projekt beinahe alle Teile eines Betriebssystems bereit, aber ein Kernel fehlte noch. Es wurde zwar immer weiter weiterentwickelt, aber man benötigte noch diverse, meistens proprietäre, Unix-Varianten, damit es lauffähig wurde. Diese Lücke eines freien Kernels wurde später durch den Linux-Kernel geschlossen.
Nachdem er 1997 seinen Aufsatz „Die Kathedrale und der Basar“, in dem er die dezentrale Entwicklungsmethode als „Basar Methode“[12] bezeichnet und sie mit der herkömmlichen „Kathedralen Methode“[13] verglich, publizierte, gründete Eric S. Raymond 1998 die Open Source Initiative. Zuvor hatte er sich dazu entschlossen, „dass die Freie-Software-Gemeinschaft ein besseres Marketing benötige“. Er wollte die freie Software als „geschäftsfreundlicher“ darstellen, da das „frei“ in „freie Software“ missverstanden werden konnte. Durch die Einführung des Begriffs „Open Source“ wollte er außerdem die freie Software als „weniger ideologisch belastet“ darstellen (Wikipedia, Open Source, Abschnitt: Geschichte). „Ende der 1990er Jahre gewann Open Source Software [schließlich] immer mehr an Aufmerksamkeit und wurde von zahlreichen Software- und Hardwareanbietern (IBM, HP) unterstützt sowie

[11] Schwalbe, 2.3.3
[12] Basar, da hier alle interessierten Entwickler mitarbeiten können.
[13] Sprich die Software wird nach einem zentralen Plan entwickelt.

von öffentlichen Einrichtungen gefördert.“[14]

4.3 Linux

Die Geschichte des Linux-Kernels beginnt 1991 in Helsinki mit Linus Torvalds, welcher eigentlich nur an einer Terminalemulation arbeitete. Dieses Programm war hardwarenah und lief ohne Betriebssystem auf Grundlage des Minix-Systems und des GNU-C-Compilers. Während der Entwicklung merkte er, dass er eigentlich ein Betriebssystem geschrieben hatte und teilte am 25.August 1991 in einem berühmten Usenet-Posting sein Programm mit einer Minix-Gruppe. In diesem Posting wird klar, dass er damals noch nicht das Ausmaß erahnen konnte, dass seine Arbeit haben würde. So bezeichnete er sein Projekt nur als „ein Hobby“, welches nicht „groß und professionell sein [wird] wie GNU“(Wikipedia, Geschichte von Linux, Abschnitt: Entstehung des Linux-Kernels). Auch glaubte er, dass es nie portierbar sein würde, etwas, was heute nicht mehr zu trifft, da Linux der meist portierteste Kernel ist. Am 17.September 1991 wurde die erste Linux-Version, 0.01, auf einem FTP-Server hochgeladen, „[um] anderen Leuten die Möglichkeit zu geben, am System mitzuarbeiten oder Verbesserungsvorschläge zu machen“[15]. Der Name „Linux“ kommt daher, dass der für den Server Verantwortliche, Ari Lemmke, den Bereich nicht, wie von Torvalds vorgeschlagen, „Freax“ nennen wollte. Er hat ihn einfach Linux genannt und Torvalds hat dies akzeptiert.
Zu Beginn stand der Linux-Kernel noch unter einer von Torvalds formulierten proprietären Lizenz. „Er merkte jedoch bald, dass das den Fortschritt der Entwicklung behinderte. Er wollte allen Entwicklern deutlich mehr Freiraum geben“[16], darum kündigte er im Januar 1992 an, den Linux-Kernel unter die GNU GPL zu stellen. Dies geschah Mitte Dezember des selben Jahres mit Version 0.99 des Kernels. Nun war es möglich, Linux mit Hilfe von GNU als ein freies Betriebssystem zu veröffentlichen. Dadurch wurde viele Entwickler neugierig, da sie nun das System einfacher modifizieren und verbreiten konnten, so dass 1993 schon über 100 Entwickler am Linux-Kernel arbeiteten, was die Veröffentlichung der ersten Linux-Distribution ermöglichte. „Später sagte Linus Torvalds in einem Interview, dass die Entscheidung, Linux unter die GNU GPL zu stellen, die beste gewesen sei, die er je getroffen habe: ’Making Linux GPL’d was definitely the best thing I ever did.’“(Wikipedia, Geschichte von Linux, Abschnitt: Linux unter der GNU GPL)
Allerdings dauerte es noch bis noch bis zum März des nächsten Jahres, bis eine, laut Torvalds Meinung, ausgereifte Version 1.0 des Kernels veröffentlicht wurde. Diese war erstmals netzwerkfähig. Von da an ging es schnell weiter. 1996 wurde Version 2.0

[14] Schwalbe, 2.3.3
[15] Wikipedia, Geschichte von Linux, Abschnitt: Der Name Linux
[16] Wikipedia, Linux, Abschnitt: Historische Entwicklung

veröffentlicht: „Der Kernel [konnte] nun mehrere Prozessoren gleichzeitig bedienen und wird damit für viele Unternehmen eine ernstzunehmende Alternative in vielen Arbeitsbereichen“[17]. Im selben Jahr wurde auch Tux als Linux-Maskötchen festgelegt. 1998 began die Arbeit an der GUI KDE und „Viele namhafte Unternehmen wie IBM, Compaq und Oracle kündigen ihre Unterstützung für Linux an“[17]. 1999 wurde Linux 2.2 veröffentlicht und die Arbeit an der graphischen Benutzeroberfläche GNOME began. Im Jahre 2000 kam ein wichtiges Programm für Linux dazu, die heute als Open Office bekannte Office-Suite StarOffice. 2001 wurden mit der 2.4er-Serie „bis zu 64 Gigabyte Arbeitsspeicher, 64-Bit-Dateisysteme, USB und Journaling-Dateisystem“[17] unterstützt. Ende 2003 folgte schließlich der Linux-Kernel 2.6. Ein weiteres wichtiges Ereignis war die Gründung der Linux Foundation 2007, welche Linux bis heute fördert. Dort hat auch Linux Torvalds ein Festanstellung. Die aktuelle Kernel-Version ist 3.19 (Stand 15.02.2015).

5 Nutzen für die Beteiligten

Das Open Source Konzept hat Nutzen für mehrere Beteiligte. Um dies besser unterteilen zu können, wird im Folgenden einzeln auf diese eingegangen. Dabei kann zu Überschneidungen kommen, vor allem bei Verbrauchern und Entwicklern. Während der Darstellung wird auch häufig auf den Wikipediaeintrag über „Die Kathedrale und der Basar” von Raymond verwiesen (im folgenden kurz einfach Raymond).

5.1 Nutzen für das Projekt

Bei der Entwicklung der Open Source Software ist die dabei entstehende Software eigentlich das Wichtigste. Das Open Source Konzept hilft dabei in sehr großem Maße. Prof. Dr. Schwalbe meinte dazu in Kapitel 5.1 seiner Arbeit: „Um jedoch eine qualitativ hochwertige Software bereitzustellen, bedarf es eine hohe Anzahl an Programmierern.“ Durch die Veröffentlichung des Quelltextes ist es jedem möglich, der mitarbeiten kann, zu dem Projekt beizutragen, wodurch die Software schneller besser wird. Das sieht man auch an der Entwicklung des Linux-Kernels. Kaum hatte Torvalds das Projekt unter die GPL gestellt, ging die Entwicklung sehr viel schneller voran. Laut einer Angabe auf Wikipedia[18] entstanden 2007 täglich durchschnittlich „4.300 Zeilen neuer Code“. Damit nicht der Überblick über das Projekt verloren geht und damit keine „schlechten“ Codezeilen in das offizielle Projekt aufgenommen werden, gibt es bei den Projekten immer noch die so genannten „Maintainer“, die entscheiden, was in die offizielle Version aufgenommen wird. Bei Linux sind dies allen

[17] Wikipedia, Geschichte von Linux, Abschnitt: Chronologie
[18] Wikipedia, Linux, Abschnitt: Entwicklung

voran Linus Torvalds und einige andere. Bei größeren Projekten wird noch einmal in Subsysteme unterteilt. Jeder Maintainer bemüht sich dabei, das Beste für sein Teilprojekt zu erreichen (nach der Basar-Methode von Raymond). Im Gegensatz zur Kathedralen-Methode ist eine solche nach der Basar-Methode entwickelte Software, nach Raymond, nie „fertig“. Da sie keinem festen Plan folgt, wird sie einfach entwickelt und immer weiter verbessert. Bei proprietärer Software kommt hingegen oft die Kathedralen-Methode zu tragen. Irgendwann wird einfach gesagt, dass die Software fertig sei. Die hat den Nachteil, dass nicht mehr weiterentwickelt wird. Bei den meisten Open Source Projekten hingegen gibt es eine stetige Weiterentwicklung. Dies garantiert die Aktualität der Software, da alle neuen Probleme, die auftreten[19], sofort gelöst werden können. Der letzte Punkt, den ich hierzu nennen möchte, ist der Vorteil der schnelleren Problemlösung. Nach der 8. Richtlinie von Raymond findet man bei einer großen Gruppe von Entwicklern schneller eine Lösung als bei einer kleineren.
Zusammengefasst nutzt Open Source dem Projekt in folgenden Punkten: 1. schnellere und qualitativ bessere Entwicklung, 2. ständige Aktualität und 3. schnellere Problemlösung.

5.2 Nutzen für den Verbraucher

Der Verbraucher ist, nach dem Projekt an sich, der wichtigste zu beachtende Teilaspekt, da die Software für diesen gemacht wird und deshalb auch dessen Bedürfnissen entsprechen sollte. Zuerst einmal hat der Benutzer durch die Veröffentlichung des Quelltextes die Möglichkeit, die Software zu prüfen und kann so effektiv entscheiden, ob er die Software nutzen will oder nicht. Sollte es ihm nicht ganz gefallen, so steht es ihm frei, die Software zu erweitern oder an seine Zwecke anzupassen wie er möchte[20]. Dabei kann er auch frühere Versionen verwenden, da diese größtenteils gespeichert bleiben. So werden die verschiedenen Versionen des Linux-Kernels auf kernel.org archiviert, was einen Vorteil darstellt, da es teilweise große Unterschiede zwischen den Versionen gibt, so dass nicht unbedingt die neuste Version auf einem Computer läuft. Ein ziemlich großer Vorteil für den Verbraucher (und auch für das Projekt an sich) ist die Möglichkeit der Mitwirkung. Dies ist zwar auch bei proprietärer Software möglich, aber nicht in so einem Maße wie bei Open Source Software. Hier kann der Verbraucher schon während des Entwicklungsprozesses Rückmeldung geben und sogar eigenen Ideen einbringen. Dazu sagte Raymond: „Fast so gut wie eigene gute Ideen zu haben, ist es, gute Ideen von den Benutzern zu erkennen. Manchmal ist

[19] vgl. mit der 6. Richtlinie von Raymond

[20] Dies ist bei diverse proprietärer Software auf Grund von Lizenzen untersagt. Dabei ist auch oft der Verwendungszweck eingeschränkt.

letzteres besser“[21]. Die eigenen Ideen der Verbraucher sind insofern besser, da sie den Bedarf widerspiegeln. Der zweite große Nutzen für den Verbraucher ist die im Vergleich zu proprietärer Software relativ schnelle und hochwertige Hilfe. Zum einen werden die Programmfehler bei Meldung relativ schnell behoben, da mehrere Personen daran arbeiten und der Benutzer vorarbeiten kann, zu anderen kann diesem bei Problemen von sehr viel mehr Leuten geholfen werden als wenn nur wenige Personen daran arbeiten würden. Dies wird auch in dem anschließenden Fallbeispiel deutlich. Insgesamt nutzt das Open Source Konzept dem Verbraucher in folgenden Punkten: 1. die Möglichkeit, die Software zu prüfen, 2. die Möglichkeit, diese anzupassen, 3. der großen Möglichkeit der Mitwirkung und 4. der schnellen Hilfe bei Problemen. Dabei tragen Punkt 3 und 4 gleichzeitig zur Verbesserung der Software bei.

5.2.1 Ein Fallbeispiel

Vor ungefähr einem halben Jahr hatte ich nach dem Update des Linux-Kernels auf die Version 3.17.4 auf meinem Laptop das Problem, dass mein Bildschirm nach dem Aufwachen aus dem Bereitschaftsmodus schwarz blieb und nichts mehr reagierte. Die Archievierung der Kernel-Versionen ermöglichte es mir, auf die Version 3.17.1 zu downgraden. Da das Aufwachen von da an wieder normal funktionierte, wusste ich, dass es einen Bug im Kernel gab. Durch Suchen auf diversen Foren, wo der Programmfehler schon berichtet worden war, fand ich eine Teilzeitlösung: Ich musste dem Kernel bei Boot einen zusätzlichen Parameter mitgeben: „dis_ucode_ldr“. Da dieses „workaround“ funktionierte, konnte ich meine Kernelversion wieder Updaten ohne das dieses Problem auftrat. Innerhalb kürzester Zeit gab es auch einen Patch für den Kernel. Mit der Kernel-Verion 3.17.8 war der Fehler schließlich behoben, so dass der Parameter nicht mehr notwendig war[22].

5.3 Nutzen für die am Projekt teilnehmenden Entwickler

Da die Entwickler in besonders großem Maße am Erfolg des Projektes beteiligt sind, lohnt es sich sie sich genauer zu betrachten. Der Nutzen für die teilnehmenden Entwickler lässt sich in zwei Bereiche unterteilen. Zum einen, welchen Nutzen haben sie privat daran, sich in einem solchen Projekt zu engagieren[23], zum anderen ist da noch der wichtigere Bereich, bei welchem es um die Vorteile beim Entwickeln an sich geht.
Da der erste Teilbereich nicht Hauptthema der Arbeit ist, wird er nicht so ausführ-

[21] 9. Richtlinie von Raymond, vgl. auch mit der 7.
[22] Dieses Forum hat mir damals weitergeholfen: `https://bugs.archlinux.org/task/42820`
[23] Schließlich werden 50% der Programmierer (Schwalbe, 6) für ihre Open Source Aktivitäten nicht bezahlt. Warum die anderen 50% von Unternehmen bezahlt werden ist Teil des nächsten Kapitels.

lich behandelt wie der zweite. Ein mögliches Motiv für das Engagement in einem Open Source Projekt ist z. B. der Eigenbedarf. Wenn man die Software benötigt, dann schreibt man sie sie sich einfach. Ein anderes mögliches Motiv wäre die einfache Freude an der Arbeit, vielleicht auch noch der Gedanke, etwas für die Gemeinschaft zu erreichen. Ein weiteres Motiv wäre der Gedanke, dabei etwas zu lernen. „Die Entwickler müssen sich [...] nicht mit unternehmensspezifischen Anforderungen auseinandersetzen. Das bedeutet, dass sie sich allein auf die Programmierung konzentrieren können und somit ihre allgemeinen Programmierkenntnisse erweitern, ohne sich unternehmensspezifisches Wissen aneignen zu müssen“(Schwalbe, 5.2). Die allgemeinen Programmierkenntnisse zu erweitern hilft dem Entwickler im weiteren Verlauf seiner Arbeit dabei, komplexere Programme zu schreiben und auch mal in ein anderes Projekt zu wechseln. Das letzte mögliche Motiv wäre die Verbesserung seines eigenen Status. Sei es der intern in der Gruppe oder auch extern als eine Signalmotivation, wo man Firmen zeigen möchte, wie gut man programmieren kann, und auf Grund dessen eingestellt werden möchte. Dies funktioniert bei einem Open Source Projekt besser als bei einem proprietären Projekt, da bei diesem der Anteil des Entwicklers nicht nachgeprüft werden kann.[24]

Der erste Vorteil beim Entwickeln für den Entwicklern ist die große Menge an Mitprogrammierern, welche im helfen können. Auch die Betrachtung der Benutzer als Mitprogrammierer hilft dem Entwickler die Qualität seiner Arbeit zu steigern[25]. Den nächsten Vorteil beschreibt Prof. Dr. Schwalbe in Kapitel 5.2 seiner Arbeit: „Der Open Source Programmierer ist sein eigener Chef, das heißt, er trägt die vollständige Verantwortung für den Erfolg des Teilprojektes. [...] Die Leistung eines Programmierers kann daher bei Open Source Entwicklungen besser sein als bei kommerziellen Entwicklungen“. Anders als bei proprietärer Software hat der Entwickler bei einem Open Source Projekt die Möglichkeit, die Software nach seinen Vorstellungen zu entwickeln. Dies führt erstens zu einem Motivationsschub und zweitens ermöglicht es eine allgemeinere Programmierung, sprich man kann die Kompatibilität auch zu anderer als von einem Unternehmen gewünschter Hardware/Software herstellen. Ein anderer Vorteil wurde schon früh in der Computer-Geschichte offensichtlich (siehe dazu Kapitel 4.1). Es war einfach einfacher die Software zu modifizieren und zu verbreiten. Man konnte auch Teilprojekte zur Anpassung an spezielle Gegebenheiten schaffen. Ein Beispiel dafür ist die frühe Entwicklung des Unix-Betriebssystems. Der letzte Vorteil, den ich nennen möchte, hängt mit der Offenheit des Quelltextes zusammen. Dazu sagt die 2. Richtlinie von Raymond: „Gute Programmierer wissen, was sie schreiben müssen. Brillante wissen, was sie neuschreiben müssen (und was sie wiederverwenden können)“. Es ist von Vorteil, wenn bereits Quellcode vorhanden ist

[24]Für eine genauere Darlegung der Motive siehe Schwalbe, 4 und 5.

[25]vgl. mit 6. Richtlinie von Raymond

und man nicht etwas erneut entwickeln muss, was schon jemand anders entwickelt hat. Das Teilen von innovativem Programmcode beschleunigt die Entwicklung anderer, diesen Programmcode nutzender Projekte. „Auch in verschiedenen Programmen kann der gleiche Code verwendet werden. Dies umfasst Programmfunktionen, die immer wieder verwendet werden, wie zum Beispiel „Login"-Funktionen, Datenbankanbindungen, Formulare oder Dateiverwaltungen. Diese häufigen Funktionen werden in so genannten Bibliotheken zusammengefasst und bereitgestellt" (Schwalbe, 2.5.1). Da proprietäre Software ihren Quelltext geheimhällt, müssen andere Projekte diesen Programmcode noch einmal selber entwickeln, was die Softwareentwicklung sehr stark bremst. Hilfreich für die schnellere Entwicklung kann auch das Archivieren älterer Versionen sein, da diese sich manchmal, wie z. B. beim Linux-Kernel, stark voneinander unterscheiden.
Aufgelistet bietet das Open Source Konzept dem Entwickler bei seiner Arbeit folgende Vorteile: 1. die große Hilfe auf Grund der vielen Mitprogrammierer, 2. die Eigenständigkeit bei der Entwicklung, 3. die Möglichkeit zu Modifizieren und zu Verbreiten und 4. die Möglichkeit, schon vorhandenen Programmcode wiederzuverwenden.

5.4 Nutzen für Firmen

Firmen, die teilweise proprietäre Software vertreiben, unterstützen häufig Open Source Projekte. Z. B. wird Linux von IBM, HP, Red Hat und vielen anderen Unternehmen durch „Hardwarespenden an Entwickler über Treiber und Geldspenden für Stiftungen, die sich mit Linux-Software beschäftigen, bis hin zur Anstellung von Programmierern beim Unternehmen selbst"[26]. Sogar das Unternehmen Microsoft, welches zuvor noch gesagt hatte: „Linux is a cancer."(Linux ist ein Krebs), ist 2011 „the fifth largest code contributor to the Linux kernel"(ZDNet). Also, was bringt Unternehmen dazu, sich an Open Source Projekten zu beteiligen? Was nützt ihnen das?
Da Antwort ist relativ einfach. Diese Unternehmen nutzen diese Software bzw. sind auf sie angewiesen. So laufen 20% von den Betriebssystem, die Microsofts „Azure" nutzen, unter Linux (Quelle: ZDNet). Google nutzt Linux als Betriebssystem für ihre Server. Ein weiterer Grund ist der Wunsch, einen Standard zu schaffen. Man möchte nicht, wie zu Beginn bei Unix, dass die Projekte in mehrere Projekte zerfallen. Dies wäre sehr kostenintensiv, da man diese alle Unterstützen müsste, um erfolgreich zu bleiben. Ein letzter hier genannter Vorteil ist, dass man durch Beobachtung von und Beteiligung an Open Source Projekten neue Ideen sammeln kann, die man in eigenen Projekten verwerten kann. Auch gibt dies einem die Möglichkeit,

[26]Wikipedia, Geschichte von Linux, Abschnitt: Unternehmen

gute Entwickler zu entdecken und anzuwerben. Dies treibt die Entwicklung eigener Projekte weiter voran.
Schlussendlich gibt folgende Gründe für die Beteiligung von Unternehmen: 1. die Angewiesenheit, 2. Standardisierung und 3. die Möglichkeit, Ideen und Entwickler für sich zu entdecken.[27]

6 Kritik an Open Source / Probleme

In diesem Kapitel möchte ich auf ein paar Probleme eingehen, die Open Source Projekte haben. Dazu werde ich im ersten Teil Probleme nennen und im zweiten mich mit einem Artikel zu einer Studie auseinandersetzen.
Ein Problem von Open Source Software, welches Wikipedia im Unterartikel Probleme des Artikels Open Source anführt, ist, dass manche Firmen Open Source Software zwar weiterentwickeln, diese aber „so weit an eigene Systeme anpassen, dass sie praktisch nicht mehr anders verwendet werden können“. Unter anderem wird dabei die Hardware so entwickelt, dass nur die vom Hersteller signierte Software darauf verwendet werden kann. Oder man macht den Quelltext absichtlich unleserlich. Dies ist zwar tragisch, aber nur schwer zu vermeiden. Man hat letztendlich nur die Möglichkeit, andere Hardware zu benutzen oder auf die Weiterentwicklung zu verzichten bzw. sie selber zu programmieren. Dadurch ist so etwas ähnlich vorteilhaft wie ein nicht veröffentlichter Quelltext. Ein weiteres im selben Unterartikel erwähntes Problem schilderte Niklaus Wirth: „Die Open-Source-Bewegung ignoriere und behindere die Vorstellung, komplexe Softwaresysteme basierend auf streng hierarchischen Modulen aufzubauen. [...] Sie sollten rein auf die Spezifikationen der Schnittstellen der Module vertrauen. Wenn, wie bei Open-Source, der Quelltext der Module vorhanden ist, führe das automatisch zu einer schlechteren Spezifikation der Schnittstellen, da ja das Verhalten der Module im Quelltext nachlesbar ist.“ Da mir die Problematik nicht bekannt ist, kann ich hierzu nicht viel sagen. Aber diese Aussage erscheint mir eher nach einem Wunschdenken formuliert. Wenn so etwas nicht von den Entwicklern gewünscht ist, wird es halt anders gemacht. Ein Open Source Projekt wird nach der Basar-Methode organisiert, sprich es organisiert sich quasi selbst. Ein interessantes Problem nennt der Artikel zur Freien Software auf Wikipedia in seinem Trusted Computing seines Unterartikels Gefahren für freie Software. Dieses Problem behandelt die Verschlüsselung von Daten. So etwas wird meistens durch Zertifizierung in binärer Form erreicht. Bei Open Source Projekten ist aber der Quelltext offen zugänglich, so dass der Benutzer die Verschlüsselungen theoretisch umgehen kann. Dies wäre meines Erachtens nach ein Punkt, wo der Quelltext nicht zugänglich sein sollte, da hier außerdem kein nennenswerter Schaden bei der

[27] Für weitere Gründe siehe Schwalbe, 6

Softwareentwicklung entstehen würde.
Das größte Problem für Open Source Software sind aber proprietäre Schnittstellen. „Hardware-Hersteller gehen immer mehr dazu über, die Schnittstellenspezifikationen geheim zu halten, um der Konkurrenz die Nachahmung technischer Lösungsansätze zu verwehren. Der Grund hierfür liegt im zunehmenden Wettbewerbsdruck und darin, dass es billiger ist und schneller geht, einen solchen Schutz technisch einzubauen, als das errungene geistige Gut durch ein Patent für sich zu reservieren“[28]. Dabei müssen diese Unternehmen darauf achten, dass Kunden auch Betriebssysteme wie Linux benutzen. Diese müssen sie natürlich auch bedienen. Deshalb werden häufig proprietäre Treiber bereitgestellt. Solche sogenannten „Binärtreiber (Gewähren von Binärdateien ohne Offenlegung des Quellcodes) [werden] im Linux-Umfeld kontrovers diskutiert: Während manche für einen Ausschluss proprietärer Kernel-Module plädieren, befürworten andere, dass einige Hersteller überhaupt – zur Not auch proprietäre – Treiber bereitstellen“[29]. Dieses Problem, zwangsläufig Binärdateien einbauen zu müssen um Hardwareunterstützung gewährleisten zu können, ist meiner Ansicht sehr gravierend. Dies ist der Punkt, an dem die Gedanken von proprietärer Software auf die von freier Software aufeinanderprallen. Hierbei würde ich als Benutzer aber eher für die Aufnahme von Binärtreibern stimmen, da ich es lieber habe, dass die Hardware unterstützt wird, als das ich den Quelltext dieser Treiber einsehen kann (Wozu sollte ich diese auch verändern wollen?). Dieses Problem ist eher ein ethisches Problem. Die Kritik des GNU-Projektes an Open Source bezieht sich auch weitgehend auf sozial-ethische Aspekte welche für die Softwareentwicklung unerheblich sind[30].
Im folgenden werde mich mit einer Studie von Michelle Levesque zum Thema Open Source Software auf Basis eines bei „heise online“ erschienenen Artikel auseinandersetzen. Dabei kritisiert sie verschiedene Probleme: „Dies seien die Benutzeroberfläche, die Dokumentation, funktionszentrierte Entwicklung, selbstbezogene Programmierung und ’religiöse Verblendung’“[31]. Sie kritisierte diese Punkte auf Basis eines nicht genannten Open Source Projektes. Bei der Benutzeroberfläche bemängelte sie die mangelnde Möglichkeit, diese intuitiv zu verstehen. Sie bezeichnete sie als „Alptraum“. Sie meint, dass man sich mit KDE oder GNOME weniger gut zurechtfinden kann, als mit der Oberfläche von Mac OS X. „Sie [(die Tester)] hätten sich darüber gewundert, warum sie sich mit einer derart ’klobigen’ Oberfläche beschäftigen sollen, wenn Windows XP wesentlich besser funktioniere.“[32] Zuerst einmal frage ich mich, wie sie von diesem „nicht näher benanntes Open-Source-Projekt“ auf KDE

[28] Wikipedia, Freie Software, Abschnitt: Proprietäre Schnittstellen
[29] Wikipedia, Linux, Abschnitt: Hardwareunterstützung
[30] Dazu: Stallman, Warum Open Source das Ziel von Freie Software verfehlt
[31] Wilkens
[32] Wilkens

und GNOME kommt. Außerdem kann man sich als Tester schon anstellen. KDE und GNOME werden schließlich von vielen Personen genutzt, welche anscheinend damit zurechtkommen. Es ist auch eine Frage, wie die Oberfläche konfiguriert wurde und wie viel Zeit man sich damit gelassen hat. Wenn man die Oberfläche von Windows XP schon viele Jahre benutzt und damit gestartet hat, dann ist es klar, dass sie einem auf Grund der Bekanntheit besser vorkommt. Ich glaube außerdem nicht, dass man sich ohne weiteres Vorwissen auf der Windows XP Oberfläche einfach bewegen kann. Von daher erscheint mir diese Aussage zu sehr von Meinungen und Vorurteilen geprägt zu sein. Bei der Dokumentation bemängelte sie, dass sie „nur für Spezialisten verständlich sei“[33]. Bei den meisten Projekten besteht die Möglichkeit, auf Foren nachzufragen. Zusätzlich dienen Softwaredokumentationen primär dem Ziel, die Entwicklung der Software zu dokumentieren bzw. z. B. bei Java eine Befehlsreferenz zu geben. Natürlich ist diese für einen Laien nicht unbedingt verständlich und sollte nicht mit einem Benutzerhandbuch verwechselt werden. Das dritte Problem war das Fehlen einer „funktionszentrierten Entwicklung“ wie bei der Kathedral-Methode. Sie bemängelt dabei im Endeffekt, dass ihr ein Open Source Projekt nicht zielgerichtet genug vorgingen. Dies führt aber auch, nach Raymonds 14. Richtlinie, „zu Verwendungszwecken, an die [man] niemals gedacht [hätte]“. Dabei wird das zentrale Ziel aber nicht aus den Augen gelassen und da so ein Projekt nie als „fertig“ bezeichnet werden kann, ist eine „funktionszentrierte Entwicklung“ nicht notwendig. Der vierte Kritikpunkt, die „selbstbezogene Programmierung“, ist insofern richtig, dass viele Projekte aus dem einfachen Bedarf heraus entstehen und man sich so erstmal auf die Erfüllung der eigenen Bedürfnisse konzentriert. Zu Anfang war Linux auch nur eine Terminal-Emulation. Erst hinterher wurde es eine Projekt, welches sich auch an die Gemeinschaft richtete. Da bei der Open Source Programmierung der Benutzer mit einbezogen wird, ist dieser Kritikpunkt hinfällig. Der fünfte und letzte Punkt ist schließlich nur noch ein sozial-ethischer Diskussionspunkt. Insgesamt finde ich das Ergebnis der Studie, dass „der größte Teil der Computernutzer künftig auf Open-Source-Software verzichten“[34] würde, für sehr weit hergeholt. Hier prallen wiederum einfach nur zwei feste Vorstellungen aufeinander.

[33] Wilkens
[34] Wilkens

7 Schlussteil

Die Geschichte von Open Source, vor allem die des Linux-Kernels, hat gezeigt, dass Open Source als Softwareentwicklungskonzept eine schnelle Entwicklung von Software erreichen kann. Open Source schließt geistig an die anfängliche Hacker-Kultur an. Ein Projekt unter eine „freie“ Lizenz zu stellen hat Vorteile. Diese hatte auch Linux Torvalds erkannt, so dass er die Entscheidung, Linux unter die GPL zu stellen, als eine der besten bezeichnete. Dies führte zu schneller Entwicklung und besserer Problemlösung. Die Benutzer der Software sind nicht ausgegrenzt und können helfen, die Software nach ihren Bedürfnissen zu formen. Programmierer haben bei der Entwicklung auf Grund verfügbaren, schon vorhandenen Programmteilen, den vielen Mitentwicklern, ihrer Eigenständigkeit und der Möglichkeit, die Software „frei“ zu modifizieren und zu verbreiten es einfacher, qualitativ hochwertige Software zu erstellen. Unternehmen können durch die Nutzung von Open Source Software oft Geld einsparen, da diese häufig kostenlos ist. Dieses Geld können sie in die Weiterentwicklung dieser Open Source Software stecken, wodurch sie auch durch Aufgreifen neuer Ideen ihre eigene Software oder darauf angewiesene Software/Hardware zu verbessern. Probleme gibt es für Open Source Software lediglich dann, wenn es um benötigte Inhalte, die nur proprietär vorhanden sind, oder um die Notwendigkeit, etwas zu verschlüsseln, geht. Dann artet es oft zu einer sozial-ethischen Debatte aus, wo der Nutzen in Frage steht. Nichtsdestotrotz hat Open Source als Entwicklungskonzept viele Vorteile, welche nicht von der Hand zu weisen sind. Das häufige Vorkommen des Linux-Kernels in vielen Bereichen der Computerindustrie ist ein Indiz für den Erfolg dieses Konzepts.

8 Literaturverzeichnis

Prof. Dr. Schwalbe, Ulrich: **Open Source Software - Eine wirtschaftstheoretische Analyse**, überarbeitete Version von Fahrig, Thomas - `http://www.mafabo.de/thomas/da/oss-aktuell.php` (08.08.2008)

Wikipedia: **Open Source** - `http://de.wikipedia.org/wiki/Open_Source` (07.01.2015)

Wikipedia: **Freie Software** - `http://de.wikipedia.org/wiki/Freie_Software` (31.01.2015)

Wikipedia: **Linux** - `http://de.wikipedia.org/wiki/Linux` (14.01.2015)

Wikipedia: **Geschichte von Linux** - `http://de.wikipedia.org/wiki/Geschichte_von_Linux` (28.01.2015)

Wikipedia: **Linux (Kernel)** - `http://de.wikipedia.org/wiki/Linux_(Kernel)` (07.01.2015)

Stallman, Richard: **Warum Open Source das Ziel von Freie Software verfehlt**, deutsche Übersetzung von Gehring, Robert; Kohne,Joerg - `https://www.gnu.org/philosophy/open-source-misses-the-point.de.html` (30.10.2014)

Wilkens, Andreas: **Studie übt harte Kritik an Open-Source-Software** In: heise online - `http://heise.de/-97045` (15.04.2004)

Vaughan-Nichols, Steven J.:**Why Microsoft loves Linux** In: ZDNet - `http://www.zdnet.com/article/why-microsoft-loves-linux/` (29.10.2014)

Wikipedia: **Supercomputer** - `http://de.wikipedia.org/wiki/Supercomputer#Betriebssysteme` (13.02.2015)

Wikipedia: **Die Kathedrale und der Basar** - `http://de.wikipedia.org/wiki/Die_Kathedrale_und_der_Basar` (31.06.14)