Stefan Meretz (Juni 1999)

Linux - Software-Guerilla oder mehr?
Die Linux-Story als Beispiel für eine gesellschaftliche Alternative

Version 1.06, Letzte Änderung: 08.12.1999

Originalquelle: http://www.kritische-informatik.de/linuxswl.htm

Papierversion erschienen in: FIfF-Kommunikation 3/99, S. 12-21

Inhalt:

Linux - Software-Guerilla oder mehr?
Die Linux-Story als Beispiel für eine gesellschaftliche Alternative

Linux hat die Computerszene aufgemischt. Dabei ist "Linux" nur das Schlagwort einer Entwicklung, die im Jahre 1984 am Massachusetts Institute of Technology (MIT) begann, weil ein Drucker häufiger Papierstaus produzierte - später dazu mehr. Was zur Gründung des Projektes freier Software führte und wie die Entwicklung seit dieser Zeit bis heute verlaufen ist, werde ich im ersten Teil vorstellen. Es soll vor allem denjenigen einen Einblick in die Geschichte freier Software geben, die erst heute auf Linux gestoßen sind oder noch gar nichts darüber wissen. Keine Angst also: Ich werde Fachbegriffe und Abkürzungen erläutern. Im zweiten Teil schiebe ich einen Exkurs über die Geschichte und den aktuellen Stand der Produktivkraftentwicklung ein. Ich werde klären, was Produktivkraftentwicklung ist und warum man diesen Begriff braucht, um den gegenwärtigen Stand gesellschaftlicher Entwicklungen einzuschätzen. Im dritten Teil bringe ich die Ergebnisse des ersten und zweiten Teils zusammen. Ich hole Linux wieder hoch und diskutiere an diesem Beispiel den Stand der Produktivkraftentwicklung und die Chancen alternativer gesellschaftlicher Entwicklungswege.

1. Die Geschichte freier Software

Die Geschichte freier Software beginnt mit der Gründung des GNU-Projekts [1] 1984 am MIT durch Richard Stallman. Stallman ärgerte sich, daß ein Abteilungsdrucker über Papierstaus oder fehlendes Papier keine Meldung lieferte. Er nahm sich vor, die Druckersoftware umzuschreiben und das fehlende Feature einzubauen. Bei der Herstellerfirma Xerox fand Stallman jemanden, der den Quelltext [2] hatte, ihn jedoch nicht herausgeben durfte, da er eine Nichtweitergabe-Verpflichtung (Non-Disclosure Agreement) unterschrieben und sich damit zur Nichtkooperation verpflichtet hatte. Die Software war proprietär [3]. Schlagartig wurde klar: Die "paradiesischen Zustände" (Stallman) der Freiheit und offenen Kooperation der Anfangszeit der Computerei waren endgültig vorbei. Damals war es selbstverständlich, daß der Quelltext von Computerprogrammen frei ausgetauscht und diskutiert wurde. Es war so üblich wie eine Veröffentlichung einer Forschungsarbeit in einem anderen Bereich. Der *Begriff* der "freien Software" war damals unbekannt, denn es gab *nur* freie Software. Wie kam es zum Umbruch?

1.1. Die Vorgeschichte freier Software

Die (Vor-)Geschichte freier Software (vgl. Newman, 1999) ist eng mit dem Entstehen des Internets (vgl. Zakon, 1999) und dem Unix-Betriebssystem verbunden. Ausgangspunkt ist der "Sputnik-Schock" von 1957 als es der Sowjetunion gelang, einen Satelliten in der Erdumlaufbahn zu plazieren. Als Reaktion darauf wurde 1958 die Advanced Research Projects Agency (ARPA) innerhalb des Departement of Defense (DoD) gegründet. Aufgabe war die Einrichtung und Koordination von Forschungsprojekten mit dem Ziel der Erringung der (v.a. militär-) technologischen Vorherrschaft. Die Eisenhower-Administration ersetzte damit die alleinige Förderung ausschließlich militärischer Institutionen und Projekte durch eine breite Einbindung des wissenschaftlichen Potentials der USA in eine staatlich gesteuerte, militärisch ausgerichtete Forschungsstrategie. Unter dieser staatlichen Gesamtaufsicht war das Interesse an offenem Austausch der Forschungsergebnisse sehr ausgeprägt. Es entwickelten sich in der Folgezeit zahlreiche offene Standards, die zum Teil bis heute gültig sind.

Zum staatlichen Interesse an starken Standards kam das geringe Interesse der Computerindustrie an der Software. Computerindustrie war Hardwareindustrie, Software war Beiwerk zum Hardwareabsatz. Erst Ende der Sechziger Jahre wurde die Herstellung von Software als *Engineering-Aufgabe* "entdeckt" - und mit ihr die Krise derselben (vgl. Meretz, Rudolph, 1994). Schaffung öffentlicher Standards und Entwicklung von Software in Forschungsprojekten schufen ein Klima fruchtbarer Kooperation und des freien Austauschs von Ideen. Unter diesen Bedingungen wurden Produkte entwickelt, die Jahre später von der Computerindustrie "noch einmal" erfunden wurden. So wurden am Augmentation Research Center (ARC, gegründet 1965) des Stanford Research Institute folgende Technologien konzipiert und teilweise implementiert:

- verteiltes E-Mail und Mailinglisten (5 bzw. 7 Jahre vor ARPAnet);
- Textverarbeitung 10 Jahre vor den ersten kommerziellen Programmen;
- Maus als Eingabegerät 16 Jahre vor der Einführung durch Apple;
- Entwicklung einer "Fensterumgebung" 20 Jahre vor Microsoft;
- Konzipierung einer Hyperlink-Dokumentenstruktur 25 Jahre vor dem WWW.

Mit der Entwicklung und Verbreitung von Time-Sharing Computern (v.a. die PDP-Serie von DEC) entwickelte sich jene innovative Hard- und Software-Entwicklungskultur, die später als "Hacking" [4] bezeichnet wurde. Die PDP-10, die 1967 erschien, wurde für 15 Jahre die bevorzugte Maschine der Hacker.

Im Vergleich zum heutigen Entwicklungstempo verlief die Anfangszeit von Internet und Betriebssystementwicklung vergleichsweise langsam. 1969 entwickelte Ken Thompson (AT&T Bell Labs) das erste Unix-System für eine PDP-7. Im gleichen Jahr wurden die ersten 4 Computerknoten zum ARPAnet, dem Vorläufer des Internets, zusammengeschlossen. Erst 1984 wurde die Zahl von 1000 Knoten erreicht, 1992 die Millionengrenze, heute sind es über 50 Millionen. 1974 wurde das erste einsatzfähige Unix-System für eine PDP-11/45 an der Universität von Berkeley installiert. 1977 erschien die erste Unix-Distribution zusammen mit dem Quelltext, die 2.11BSD (Berkeley Software Distribution). Unix entwickelte sich in der Folge zum bestimmenden Betriebssystem, da es weitgehend unabhängig von der Hardware auf verschiedene Maschinen portiert werden konnte. Unix galt damit als "offenes" System, während proprietäre Betriebssysteme an eine bestimmte Hardware gebunden sind (vgl. [3]). 1984 verkaufte AT&T mit dem System V, Version 7, das erste kommerzielle Unix. Software, unabhängig von der Hardware, wurde zur Ware.

In dem Maße, in dem Software zur profitablen Ware wurde, zog sich der Staat aus den Innovationen zurück. Um die je eigene Software verwerten zu können, mußte der Quelltext dem Konkurrenten und damit auch dem User verborgen bleiben. Software war nur als proprietäre Software profitabel. Mit offenen Quellen hätte sich zum Beispiel Microsoft nie als das etablieren können, als was wir es heute ansehen: als monopolartiger Moloch. Staatsrückzug und Privatisierung von Software bedeuteten jedoch auch eine Aufweichung von Standards. So entstanden in der Folge sehr viele zu einander wenig oder gar inkompatible Unix-Versionen (AT&T, BSD, Sun, HP, DEC, IBM, Siemens etc.).

Die Konsequenzen für den universitären Forschungsrahmen waren verheerend. Wo früher freier Austausch von Ideen herrschte, wurden jetzt Forschende und Lehrende gezwungen, Kooperationen zu beschränken oder ganz zu unterlassen. Software als Ergebnis von Forschungsaktivitäten durfte nicht mehr dokumentiert werden, sobald es über proprietäre Software an Firmen oder Patente gekoppelt bzw. selbst für die Patentierung vorgesehen war. Richard Stallman beschreibt diese Situation so:

> "1983 gab es auf einmal keine Möglichkeit mehr, ohne proprietäre Software einen sich auf dem aktuellen Stand der Technik befindenden Computer zu bekommen, ihn zum Laufen zu bringen und zu nutzen. Es gab zwar unterschiedliche Betriebssysteme, aber sie waren alle proprietär, was bedeutet, daß man eine Lizenz unterschreiben muß, keine Kopien mit anderen Nutzern austauschen darf und nicht erfahren kann, wie das System arbeitet. Das ist eine Gräben öffnende, schreckliche Situation, in der Individuen hilflos von einem ?Meister? abhängen, der alles kontrolliert, was mit der Software gemacht wird."[5]

1.2. Die erste Phase freier Software

Ziel des GNU-Projekts und der 1985 gegründeten Free Software Foundation (FSF) war die Entwicklung eines freien Betriebssystems. Die geniale Leistung dieser Zeit bestand in der Erstellung einer besonderen Lizenz, der GNU General Public License (GPL) - auch "Copyleft" genannt. Die Lizenz basiert auf folgenden vier Prinzipien:

- das Recht zur freien Benutzung des Programms,
- das Recht, Kopien des Programms zu erstellen und zu verbreiten,
- das Recht, das Programm zu modifizieren,
- das Recht, modifizierte Versionen zu verteilen.

Diese Rechte werden gewährleistet, in dem die GNU GPL vorschreibt, daß

- der Quelltext frei jederzeit verfügbar sein und bleiben muß,
- die Lizenz eines GPL-Programms nicht geändert werden darf,
- ein GPL-Programm nicht Teil nicht-freier Software werden darf [6].

Die besondere Stärke der GNU GPL besteht in dem Verbot, GPL-Programme in proprietäre Software zu überführen. Auf diese Weise kann sich niemand offene Quelltexte aneignen und modifiziert in binärer Form in eigenen Produkte verwenden. Damit kann freie Software nicht reprivatisiert werden, die Freiheit bleibt gewährleistet.

Lizenz-Eigen-schaften Soft-ware-Art	Null-Preis	Freie Verteilung	Unbe-grenzter Gebrauch	Quellcode vorhanden	Quellcode modifi-zierbar	Alle Ab-leitungen müssen frei sein	Keine Vermischung mit pro-prietärer SW
Kommerziell ("Microsoft")							
Probe-Software, Shareware	(X)	X					
Freeware ("Pegasus-Mail")	X	X	X				
Lizenzfreie Libraries	X	X	X	X			
Freie Software (BSD, NPL, ...)	X	X	X	X	X		
Freie Software (GNU LGPL)	X	X	X	X	X	X	
Freie Software (GNU GPL)	X	X	X	X	X	X	X

Tab. 1: Vergleich der Lizenzarten

Bis Anfang der Neunziger Jahre waren alle wesentlichen Komponenten des GNU-Systems entwickelt. Nur der Kernel, das Herz des Betriebssystems fehlte noch. Der GNU-Kernel HURD hatte ein sehr ambitioniertes Design (Microkernel-Struktur) und kam in der Entwicklung nicht voran. So kam die vom GNU-Projekt unabhängige Entwicklung eines funktionierenden Kernels durch Linus Torvalds (Linux) 1991 gerade zur richtigen Zeit. Torvalds unterstellte Linux der GPL. Linux als Kernel und die GNU-Komponenten wurden zum GNU/Linux-System zusammengefügt - das, was wir heute schlicht als "Linux" kennen.

1.3. Die zweite Phase freier Software - ein neues Entwicklungsmodell

Wenn bisher einzelne Personen als Entwickler von freier Software genannt wurden, bedeutet das nicht, daß sie diese alleine die Programme entwickelten. Freie Software, früher wohl Software generell, wurde und wird in der Regel in Gruppen entwickelt. Eine Person ist dann für die Koordination der Entwicklung verantwortlich, meistens jedoch auch selbst ganz wesentlich an der Kodierung beteiligt. Bis 1991 war dabei die Auffassung verbreitet, daß kleinere Projekte oder Projekte, die sich gut in additive Module aufteilen lassen, von großen Gruppen entwickelt werden

können. Sehr große Projekte wie z.B. die Entwicklung des GNU-Kernels HURD würden ein kleines eingeschworenes Team erfordern, um den notwendigen Kommunikationsaufwand zu minimieren. Im GNU-Manifest, das Richard Stallman (1993) in der Entstehungsphase des GNU-Projekts schrieb, heißt es (bemerkenswert ist die Ergänzung in Klammern):

> "Ich habe sehr viele Programmierer gefunden, die bereit sind, einen Teil ihrer Arbeitszeit GNU zu widmen. [...] Wenn jeder Beteiligte einen kompatiblen Ersatz für ein Dienstprogramm schreibt und dafür sorgt, daß es an der Stelle der Originalkomponente richtig arbeitet, ... sollte das Zusammensetzen dieser Komponenten eine durchführbare Aufgabe sein (Der Kernel wird eine engere Zusammenarbeit erfordern, daher wird eine kleine Gruppe daran arbeiten.)"

Diese Vorstellung - große Projekte erfordern kleine Gruppen - war (und ist) im Bereich des Software-Engineerings als das Brook?sche Gesetz bekannt (Brooks, 1995). Es sagt voraus, daß bei einer Zunahme der Programmierer/innen um N die geleistete Arbeit ebenfalls um N steigt, die Komplexität und damit Fehlerwahrscheinlichkeit jedoch um N^2 ansteigt. N^2 entspricht dem kommunikativen Aufwand aufgrund der möglichen Anzahl der Schnittstellen zwischen den verschiedenen Programm-Modulen. Ein Projekt mit tausend und mehr Beteiligten sollte danach nur mit geringer Wahrscheinlichkeit ein stabiles Produkt zustande bringen. Diese Einschätzung deckt sich mit praktischen Erfahrungen (z.B. die Instabilität der Windows-Betriebssysteme von Microsoft), weshalb auch nach wie vor von einer Krise der Softwareentwicklung gesprochen wird (Meretz, Rudolph, 1994).

Anfang 1992 fand in der Newsgroup comp.os.minix die Tanenbaum-Torvalds-Debatte statt, die inzwischen weithin dokumentiert ist [7]. Das war die Zeit von 386-Computern unter DOS, von WordPerfect und DBASE. Das Mini-Unix Minix wurde ab 1986 von Tanenbaum für Lehrzwecke geschrieben, da AT&T ab UNIX Version 7 (von 1984) die Verwendung des Quelltextes generell untersagte. Vordergründig ging es um Betriebssystemdesign (Minix: Mikro-Kernel vs. Linux: monolithischer Kernel), dahinter spielte aber immer wieder die Lizenzfrage und die Frage des verteilten Entwickelns ein wichtige Rolle. Linux überrollte Minix aus zwei wesentlichen Gründen:

- Minix wurde kommerziell unter einer nicht-freien Lizenz vertrieben und durfte nicht beliebig kopiert und nicht im Quelltext modifiziert werden (nur "Patches" waren erlaubt); Linux erschien unter einer freien Lizenz mit der ausdrücklichen Erlaubnis Kopien zu verteilen und den Quelltext zu verändern.
- Minix wurde folglich nur von Tanenbaum entwickelt, während sich an der Weiterentwicklung von Linux jeder beteiligen konnte.

Folgendes Zitat verdeutlicht die Haltung Tanenbaums:

> "Ich denke, daß die Koordination von 1000 Primadonnas, die überall auf der ganzen Erde leben, genauso einfach ist wie als Hirt Katzen zu hüten ...
>
> Wer sagt, daß eine Menge weit verstreuter Leute an einem komplizierten Stück Programmcode hacken können und dabei die totale Anarchie vermeiden, hat noch nie ein Softwareprojekt gemanagt." (DiBona, Ockham, Stone, 1999, Appendix A)

Tanenbaum prophezeite Linus Torvalds, daß es unmöglich sein werde, die Kontrolle über die offizielle Version zu behalten, da die Mitentwickler wie fleißige Biber in verschiedene Richtungen streben werden. Torvalds antwortete darauf trocken, daß er gar nicht die Kontrolle behalten wolle!

Eric S. Raymond, selbst Hacker von freier Software, hat in seinem inzwischen klassischen Essay "Die Kathedrale und der Basar" (1997) die Situation so beschrieben:

> "Auch war ich der Meinung, daß Programme, die eine gewisse Komplexität überschritten, nach einem zentralistischen Ansatz verlangen. Ebenso glaubte ich, daß die wichtigsten Programme, worunter Betriebssysteme und sehr große Werkzeuge wie Emacs fallen, wie Kathedralen gebaut werden müssen. Von einzelnen, erleuchteten Künstlern oder einer Handvoll auserwählten Baumeistern hinter gut verschlossenen Türen zusammen gebaut, Stein um Stein (lies Zeile um Zeile), mit keinem Beta-Release, bevor die Zeit nicht endgültig reif ist.
>
> Linus Torvalds Entwicklungsstil überraschte mich:
> - veröffentliche früh und häufig
> - delegiere alles was sich delegieren läßt
> - sei offen bis zum Punkt des heillosen Durcheinanders genannt das Chaos...
>
> ...gelinde ausgedrückt.
>
> Nicht gerade was man eine ehrfurchtsvolle Kathedralen-Vorgehenweise nennen könnte. Die Linux Gemeinde gleicht schon eher einem großen plappernden Basar (...)
>
> Die Tatsache, daß die Basar-Entwicklungsmethode zu funktionieren schien, war für mich ein klarer Schock." (Raymond, 1997)

1.4. Basar-Projekte: Selbstorganisierte kollektive Softwareentwicklung

Ein gutes Projekt beginnt mit dem individuellen Interesse an der Erschaffung guter Software für einen bestimmten Zweck. Die Initiator/inn/en und Moderator/inn/en solcher Projekte heißen Maintainer. Das kann eine einzelne Person oder eine Gruppe von Leuten sein. Wird das eigene Interesse auch von anderen geteilt, dann finden sich schnell Mitentwickler/innen - eine Projekt-Community entsteht. In aller Regel startet ein Projekt nicht nur mit einer puren Idee, sondern mit einem ersten rudimentären, aber lauffähigen Programm, an und mit dem dann gemeinsam weiterentwickelt wird. Der Maintainer gibt neue Entwicklungsstände des Programms frei.

Entscheidend für den Projekterfolg sind folgende Faktoren:

- Fähigkeiten des/der Maintainer: Maintainer müssen sicher gute Softwareentwickler/innen sein, aber wichtiger noch sind ihre kommunikativen Fähigkeiten bei der Moderation des Projekts. Die Genialität liegt nicht beim Maintainer, sondern im Projekt. Maintainer entscheiden über die Aufnahme von Features und die Veröffentlichung neuer Version, sie sorgen gleichzeitig für Transparenz der Entscheidungen. Ein Maintainer gewinnt Autorität, wenn die getroffenen Entscheidungen vom Projekt mehrheitlich nachvollzogen werden. Maintainer erkennen gute Ideen und verstehen es, die Verschiedenheit der Perspektiven der Projektmitglieder als Kraft für das Projekt zu nutzen. Maintainer entwickeln nicht das Projekt, sondern lassen das Projekt *sich* entwickeln.
- Etablierung einer Projektkultur: Ob ein Projekt eine echte Community ausbildet, ist nicht steuerbar. Anders als verordnete Projekte in der Privatwirtschaft basieren freie Projekte auf einem starken gemeinsamen Eigeninteresse, nämlich dem an guter benutzbarer Software. Darüber hinaus verschafft ein erfolgreiches Projekt den Mitgliedern Anerkennung und das gute Gefühl etwas Sinnvolles in die Welt gesetzt zu haben bzw. daran beteiligt gewesen zu sein.

Maintainer und Projekt befinden sind in einer Win-Win-Situation: Das Projekt braucht einen guten Maintainer und gibt ihm jede Unterstützung, und der Maintainer braucht gute und viele Projektmitglieder, um das Projekt erfolgreich durchzuführen. Denn nur viele (und gute) Mitglieder finden viele Fehler. Sie werden nur beim Projekt bleiben, wenn der Maintainer sie ernst nimmt und ihre Vorschläge berücksichtigt. Die Win-Win-Situation ist auch der Grund dafür, daß es selten zu Abspaltung von Projekten kommt. Es gibt einen starken sozialen Druck gegen die Spaltung, denn Spaltungen gefährden den Projekterfolg. Kommt es dennoch zu Abspaltungen, was wirklich extrem selten der Fall ist, dann besteht eine starke Notwendigkeit, diesen Schritt in der Öffentlichkeit gut zu begründen, um mindestens eine Tolerierung eines solchen Schrittes zu erhalten. Eine Abspaltung ist immer mit der Neubenennung des neuen Projekts verbunden und richtet sich in aller Regel nicht gegen das Stammprojekt.

Mit den psychologischen Aspekten dieses selbstorganisierten kollektiven Entwicklungsmodells setze ich mich an anderer Stelle auseinander (Hier findest Du dann in einer späteren Version einen Link auf diesen Aufsatz).

1.5. Zusammenfassung

Linus Torvalds war der Erste, der die Potenzen des Internets als verbindendes, globales Kommunikationsmedium für die Software-Entwicklung nutzte. Er schöpfte den neuen Möglichkeitsraum voll aus. Während kommerzielle proprietäre Softwareprojekte den Profitinteressen des Unternehmens unterliegen, die Anforderungen innerhalb des Projekts folglich letztlich fremdbestimmt sind, finden sich freie Softwareprojekt auf der Grundlage des eigenen Interesses an der Sache und natürlich auch aus Spaß an der Sache zusammen.

Das Eigeninteresse im freien Softwareprojekt ist aufgrund der Struktur der Projekte völlig verschieden vom sicherlich auch in kommerziellen Projekt vorhandenen eigenen Interesse "die Sache gut zu machen". Das Eigeninteresse im freien Softwareprojekt kann sich nur mit den Anderen also in Übereinstimmung mit den Interessen anderer entfalten. Im kommerziellen Projekt besteht hingegen die starke Tendenz, sich auf Kosten anderer zu profilieren. Es ist bemerkenswert, wenn Microsoft von der freien Softwarebewegung lernen will, in dem jetzt die verschiedenen Entwicklungsteams sich gegenseitig in den Quelltext gucken dürfen!

2. Die Geschichte der Produktivkraftentwicklung

Wer keinen Begriff vom "Wald" hat, für den bleiben auch noch so viele Bäume immer nur Bäume. Wer keinen Begriff von der Produktivkraftentwicklung hat, für den bleibt die Geschichte der Menschheit nur eine Abfolge technischer Innovationen und Erfindungen.

Linux ist nicht einfach ein neues, qualitativ hochwertiges Produkt, eine neue Erfindung sozusagen, sondern Linux steht für eine neue Qualität der Produktivkraftentwicklung. Um diese These zu erläutern, ist es zunächst notwendig den Inhalt und den Sinn des Begriffs der Produktivkraftentwicklung zu verstehen.

2.1. Produktivkraft der Arbeit

Menschen arbeiten, um die Mittel für ihr Leben zu schaffen. Diese Arbeit ist unterschiedlich wirkungsvoll. Der Begriff der Produktivität ist ein Maß für die Menge der hergestellten Güter pro Zeiteinheit. Damit ist jedoch nur die Quantität erfaßt. Produktivitätsvergleiche machen eng gefaßt nur Sinn, wenn es um ein und dasselbe Produkt geht. Ich kann nicht Fabrikarbeiter und Bauer vergleichen. Ich brauche also einen Begriff für Inhalt und Art der Arbeit. Dieser Begriff ist der der Produktivkraft der Arbeit wie ihn Karl Marx im "Kapital" [8](1976, S. 54) und anderen Schriften verwendet. Mit

diesem Begriff lassen sich verschiedene Aspekte der Arbeit erfassen:

- WAS - Inhalt der Arbeit: Art der Produkte und Mittel zu ihrer Herstellung;
- WIE - Art der Arbeit: Arbeitsorganisation;
- WIEVIEL - Produktivität der Arbeit: die je Zeiteinheit hergestellte Produktmenge.

Die Produktivkraft der Arbeit umfaßt also sowohl die quantitativen wie qualitativen Aspekte der Arbeit. Mich interessiert vor allem letzteres: das WAS und WIE der Arbeit. Das jedoch nicht bloß für einen bestimmten Zeitpunkt, sondern mich interessiert das WAS und WIE der Arbeit *in seiner Entwicklung*. Das nenne ich dann "Produktivkraftentwicklung" - ein eher neuer Begriff, den es bei Marx noch nicht gab. Warum aber soll Wissen über die Entwicklung der qualitativen Aspekte der Arbeit nützlich sein - gar für das Verständnis von Linux?

Der Grund dafür ist mein Verständnis vom Wesen der Erkenntnisse. Kurz gesagt: Man versteht wie etwas ist, wenn man versteht wie es geworden ist. Geschichte ist so nicht nur eine Sammlung historischer Fakten - einzelnen "Bäumen"-, sondern wird als Erklärung für das heute Beobachtbare der Schlüssel zu Verstehen des Ganzen - der "Wald" wird sichtbar. Um das Ganze, den "Wald", verstehen zu können, ist es erforderlich, die Dynamik der Entwicklung in dieser Geschichte zu verstehen. Dafür braucht man einen analytischen Begriff, sozusagen die Brille, mit der ich auf die Geschichte gucke. Ohne einen solchen Begriff sehe ich "nichts" - oder "alles", was auf das Gleiche rausläuft.

Mein Begriff für Linux ist der der Produktivkraftentwicklung. Meine Frage ist demnach: Wie kann ich das, was wir am Beispiel von Linux beobachten können, diese besondere Art der Arbeit, in die Geschichte der Arbeit, in die Geschichte der Produktivkraftentwicklung einsortieren? Können wir diese Frage beantworten, dann können wir den Streit darüber aufklären, ob die Linux-Arbeit Elemente einer gesellschaftlichen Alternative enthält oder nicht.

2.2. Gesellschaftsformationen und Perioden der Produktivkraftentwicklung

Als ob das alles nicht schon kompliziert genug wäre, muß ich noch einen Begriff dazunehmen. Die Produktivkraftentwicklung, das werde ich gleich ausführen, sagt noch nichts über die Gesellschaft aus, in der sie stattfindet. Dafür brauche ich den Begriff der Gesellschaftsformation. Der ist aber ziemlich leicht nachzuvollziehen, da inzwischen auch die Befürworter unsere Wirtschaftsweise "Kapitalismus" nennen. Bei uns geht es ums Geld, um die Maximierung des Profits. Die Macht haben folglich die, die das am besten können, das ist das die bürgerliche Klasse, das Kapital. Historisch vor der Bürgerklasse hatten die Feudalen, die als Bestimmer über Grund und Boden andere für sich arbeiten ließen, das Sagen. Die Epoche vor dem Feudalismus nennt man Sklavenhaltergesellschaft, sie basiert auf der direkten Verfügung über Menschen gleich Tieren und dem Raub ihrer Arbeitsergebnisse. Weitere Differenzierungen [9] sowie die Urgesellschaft als "machtlose" Gesellschaft vor der Sklavenhaltergesellschaft lasse ich hier einfach mal weg.

Sklavenhaltergesellschaft und Feudalismus waren agrarische Gesellschaften. Das WAS der Arbeit ist also schnell klar: Es ging um die Herstellung von Nahrungsmitteln mit Hilfe von einfachen Werkzeugen und unter Nutzung menschlicher und tierischer Kraftanstrengung. Hier zeigen beide Gesellschaften keine großen Unterschiede. Anders beim WIE der Arbeit. Die landwirtschaftlichen Produzenten waren mehrheitlich Leibeigene ihrer Feudalherren, waren so im Unterschied zum Sklaven also nicht personaler Besitz. Trotz Abgabenzwang und Frondienste war der relative Spielraum der Fronbauern zur Entfaltung der Produktivkraft der Arbeit größer als bei den Sklaven, die - da personaler Besitz - gänzlich kein Interesse an der Steigerung der Produktion hatten. Aufgrund des höheren Mehrprodukts konnten sich Handwerk und Gewerbe rasch entwickeln.

Spannend wird es mit dem Übergang zum Kapitalismus. Hier kommt es zu einem doppelten Umschlag. Sowohl die Art und Weise der Produktivkraftentwicklung änderte sich in all ihren Aspekten als auch die gesellschaftliche Machtstruktur wurde komplett umgestülpt. Mehr noch: Die qualitative Änderung bei der Entwicklung der Produktivkräfte der Arbeit bewirkte und erforderte eine Änderung der gesellschaftlichen Formation - und umgekehrt. Der aufziehende Kapitalismus wurde zur Industriegesellschaft. Die Agrarproduktion verschwand damit nicht als wichtiger Wirtschaftszweig, bestimmend wurde jedoch die Industrie. Die Landwirtschaft wurde selbst in der Folgezeit industrialisiert. Die industrielle Revolution war nur möglich, in dem der enge feudale Rahmen gesprengt wurde. Leibeigene wurden als Lohnarbeiter für die Industrie "befreit", der Innovationen hemmende Feudaladel von der ökonomisch expandierenden bürgerlichen Klasse beiseite geräumt oder ruhig gestellt.

Zusammenfassend nach den drei Aspekten der Produktivkraftentwicklung:

- WAS: Die ersten Klassengesellschaften bis an die Schwelle zum Kapitalismus waren durch die Produktion landwirtschaftlicher Produkten bestimmt. Sie waren Agrargesellschaften. Im Feudalismus entstand Handwerk und Gewerbe als bedeutender eigenständiger Zweig.
- WIE: Die Bodenbearbeitung erfolgte mit einfachen Hilfsmitteln. Während die Arbeit der Sklaven durch unmittelbaren Zwang angeeignet wurde, hatten die vorwiegend leibeigenen Bauern trotz Zwangsabgaben und Frondiensten ein gewisses Eigeninteresse an der Verbesserung und Entwicklung der Produktion (Verbesserung der Arbeitsmittel und der Organisation der Produktion: Dreifelderwirtschaft).
- WIEVIEL: Folglich war die Produktivität im Feudalismus höher als die in den Vorläufergesellschaften. Auf Grundlage des erweiterten Mehrprodukt konnten sich Handwerk und Gewerbe entfalten.

2.3. Phasen der Produktivkraftentwicklung im Kapitalismus

Sein Wesen und grundlegenden Funktionsprinzipien hat der Kapitalismus seit Anbeginn nicht geändert, dennoch vollzog sich eine gravierende innere Umgestaltung. Auch hier wieder änderten sich nicht so sehr die Produkte und Mittel zu ihrer Herstellung (das WAS), sondern vor allem das WIE durchlief verschiedene Phasen. Sie seien hier nur zusammenfassend dargestellt. Eine ausführliche Darstellung habe ich mit dem Aufsatz "Die doppelte algorithmische Revolution des Kapitalismus" vorgelegt (Meretz, 1999). Die drei Phasen der Produktivkraftentwicklung innerhalb des Kapitalismus nenne ich in Übernahme vorhandener Begriffe Industrielle Revolution [10], Fordismus und Toyotismus - die letzten beiden, weil jeweils die Autoindustrie beispielhaft für den Entwicklungsstand des Gesamtsystems stand (und steht).

Die Entfaltung des Kapitalismus und die industrielle Revolution waren wechselseitig miteinander verknüpft. Die Durchsetzung des einen bedingte und erforderte die Durchsetzung des anderen. Fälschlicherweise wird auch heute noch angenommen, daß die Dampfmaschine die industrielle Revolution bewirkte. Dem ist nicht so. Das Wesen dieser ersten Stufe bestand in der Revolutionierung der Werkzeug- oder Prozeßmaschine, in der verallgemeinerten Übertragung der individuellen Handwerkzeuge und Techniken auf einen maschinellen Prozeß und seiner nachfolgenden Verwissenschaftlichung durch Anwendung der Naturwissenschaften (Maschinenbau, Metallurgie, Chemie etc.). Die Standortunabhängigkeit der Energie - zunächst der kinetischen, später der elektrischen Energie - beförderte diesen "Übertragungsprozeß" zwar, war jedoch nicht ursächlicher Antrieb.

Neben der Energiemaschine und der Prozeßmaschine gibt es einen dritten Bestandteil des industriellen Prozesses, den ich algorithmische Steuerung nenne. Sie verallgemeinerte das intuitive algorithmische Wissen des Handwerkers und ermöglichte eine Abbildung dieses Wissens in einem Maschinenprozeß. Locker gesprochen waren die industriellen Maschinen "festverdrahtete analoge Spezialcomputer" mit nur einem oder wenigen "Programmen". Dieser dritte Bestandteil des industriellen Prozesses, die Algorithmusmaschine, wurde Gegenstand der nachfolgenden Umwälzungen.

Der Fordismus führte die Algorithmisierung der Produktion konsequent durch. Augenfälligstes Resultat dieser Algorithmisierung war das Fließband. Die Entfernung jeglicher Reste von Subjektivität der arbeitenden Menschen aus der Produktion war Programm ("Taylorismus"). Der Mensch wurde zum vollständigen Anhängsel der Maschine, in der der von Ingenieuren vorgedachte Algorithmus des Produktionsprozesses vergegenständlicht war. Diese Produktionsweise basierte auf der massenhaften Herstellung uniformer Güter.

Die nächste Stufe, der Toyotismus, war der Versuch einer Reaktion auf die großen Absatzkrisen, die Mitte der 70er Jahre einsetzten. Konkurrenzfähig war nunmehr nicht derjenige, der massenhaft Güter sehr billig produzierte, sondern derjenige, der am schnellsten auf Marktanforderungen reagierte - gleichwohl ohne die Waren zu verteuern. Die "festverdrahteten" Produktionsprozesse der fordistischen Ära waren dafür zu unflexibel, die betrieblichen Hierarchien zu starr. Der versuchte Ausweg war eine Verlagerung der Algorithmen aus der Hardware in die Software, war also der Versuch Flexibilität selbst als Merkmal zu implementieren. In gewisser Weise spiegeln Microsoft und SAP mit ihren Produkten genau diesen Versuch wider. Flexibilität, Anpaßbarkeit und eingängige Benutzbarkeit ist "fest" implementiert. Das ist der Versuch, alle möglichen Kundenwünsche vorauszuahnen und zu realisieren. Es liegt meines Erachtens auf der Hand, daß dieser Weg eine Sackgasse darstellt.

Der Toyotismus ist in der Krise, denn auch der Weg, Flexibilität als Merkmal zu implementieren, ist noch zu starr. Auch die verschiedenen Wunder-Methoden zur Mobilisierung der Mitarbeiter-Kreativität z.B. aus Japan sind untaugliche Versuche. Es wird zwar verschiedentlich erkannt, das die letzte expandierbare Ressource der Mensch selbst ist, doch es wird nicht gesehen, daß sich diese Ressource Mensch nur *selbst* entfalten kann und *nicht von außen* "entfaltbar" ist.

Zusammenfassend nach den drei Aspekten der Produktivkraftentwicklung:

- WAS: Der Kapitalismus produziert Güter mit industriellen Mitteln. Zweck der Produktion ist der Verkauf der Güter und damit die Realisierung des Profits. Der Gebrauchswert der Güter interessiert nur als Mittel zum Zweck.
- WIE: Der Einsatz von Technik und Wissenschaft ist das Mittel, um Arbeit und Produktion zu entwickeln. Der Fordismus versucht unter Einsatz dieses Mittels den subjektiven Faktor aus der Produktion auszuschließen, während der Toyotismus ihn wieder reintegrieren will.
- WIEVIEL: Die Produktivität konnte mit fordistischen Mitteln gegenüber der Phase der Industriellen Revolution beträchtlich gesteigert werden. Quelle war die Revolutionierung des algorithmischen Produktionsaspektes. Der Toyotismus versucht die Quadratur des Kreises: Durch Algorithmisierung der Algorithmisierung, durch Festlegung des Flexiblen, durch Vorschreiben von Kreativität sollen die Nachteile des Fordismus aufgehoben werden.

2.4. Ein neuer Qualitätssprung in der Produktivkraftentwicklung steht an

Die nachfolgend dargestellte Abbildung dampft das Gesagte noch einmal beträchtlich ein. Anhand der Übersicht wird deutlich, wo wir heute stehen: an der Schwelle zu einem neuen Qualitätssprung sowohl in Produktivkraftentwicklung als auch Gesellschaftsformation.

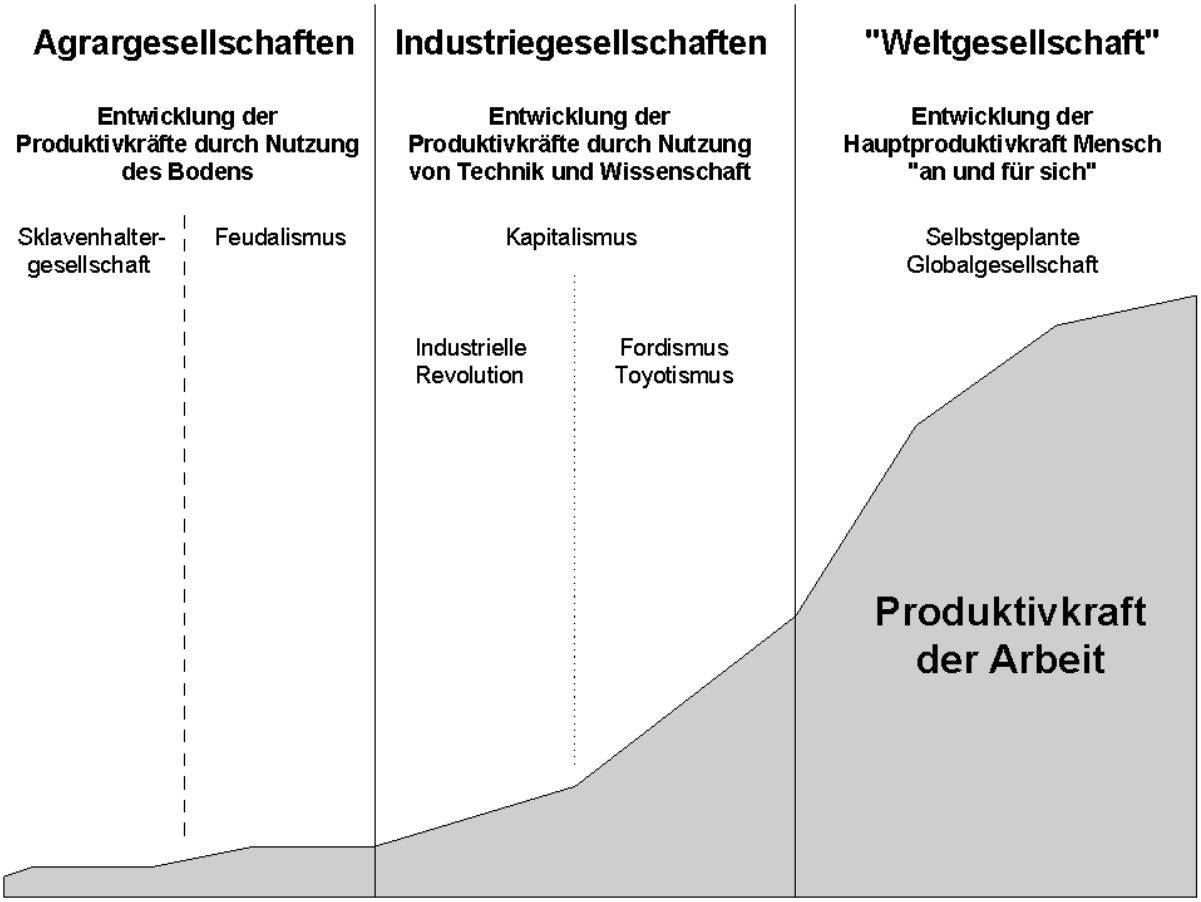

Bild anklicken, um es vergrößert anzuzeigen. Danach im Browser auf "zurück" klicken.

Abb. 1: Die drei Epochen der Produktivkraftentwicklung

Die neue "Weltgesellschaft" wie ich sie mal genannt habe, zeichnet sich durch folgende Merkmale aus:

- sie ist global und hoch vernetzt
- sie baut auf Industrie- und Agrargesellschaften auf
- sie wird durch einen neuen Typ der Produktivkraftentwicklung bestimmt
- sie funktioniert nach dem "Basar-Modell": dezentral selbstorganisiert und selbstgeplant

Das sind heftige Thesen, die der Erläuterung bedürfen. Zunächst: Meine Vorstellung bezieht sich nicht bloß auf den kleineren Teil der Menschheit, der in den sogenannten entwickelteren Zonen der Erde lebt. Es ist ein globales Modell, das gleichwohl Unterschiede auf einem angenäherten Niveau der Lebenschancen der Menschen zuläßt.

Der erste Spiegelstrich dürfte die geringsten Kontroversen hervorrufen. Schon heute können wir Globalisierung und Vernetzung als markante Tendenzen beobachten. Daß Agrarwirtschaft und industrielle Produktion von Gütern weiterhin stattfinden wird, ist vermutlich auch klar. Kontrovers ist sicherlich, daß Technik und Wissenschaft nicht mehr der bestimmende Entwicklungsmodus der Produktivkraft der Arbeit sein wird. Ich habe den neuen Modus "Entwicklung der Hauptproduktivkraft Mensch an und für sich" [11] genannt. Was ist damit gemeint?

Die Wendung "an sich" und "für sich" und "an und für sich" kommt von Hegel und bedeutet verkürzt folgendes [12]:

- "An sich" kann man übersetzen mit "der Möglichkeit nach". Im Zusammenhang hier ist das der Möglichkeitsraum des Gattungswesens Mensch. Dieser Raum der Möglichkeiten wird durch die gesellschaftliche Natur des Menschen bestimmt. "An sich" hat der Mensch alle Möglichkeiten, nur...
- "Für sich" oder übersetzt: "der realisierten Form nach" kann er seine Potenzen nicht entfalten, da einer Entfaltung zahlreiche Einschränkungen entgegenstehen. Diese Einschränkungen resultieren aktuell aus der kapitalistischen Grundstruktur unserer Gesellschaft, die durch Konkurrenz und das Durchsetzen auf Kosten anderer statt globaler Kooperation und gemeinsamer globaler Entwicklung bestimmt ist.
- "An und für sich" bedeutet schließlich die Aufhebung des Widerspruch zwischen Möglichkeit und Realisierung. Der Mensch entfaltet seine Gattungspotenzen in globaler Kooperation.

Die unbeschränkte Entfaltung des Menschen, also das Fallenlassen aller Verstümmlungen, Behinderungen und Einschränkungen, die der Kapitalismus für die Menschheit bedeutet, setzt einen dramatischen Entwicklungsschub der Produktivkraft der Arbeit frei. Das bedeutet, daß die Produktivkraft der Arbeit nicht mehr vorherrschend durch Technik und Wissenschaft (der "alten" Weise) gesteigert wird, sondern durch die Selbstentfaltung des Menschen in globaler Kooperation mit anderen. Damit ist auch die "Selbstentfaltung" und die "Entfaltung der Anderen" kein Widerspruch mehr - wie wir das jetzt unter unseren Bedingungen erleben. Im Gegenteil, das Verhältnis kehrt sich geradezu um: Für meine Selbstentfaltung ist die Entfaltung der Anderen die Voraussetzung - und umgekehrt. Selbstentfaltung als Selbstzweck zum Nutzen aller!

Was das ökonomisch bedeutet, habe ich an anderer Stelle versucht zu beschreiben (Meretz, 1999). Nur soviel sei skizziert: Eine Globalgesellschaft kann nicht zentral geplant und verwaltet werden, die bekannten Versuche sind ja auch gescheitert. Sie wird sich "selbst planen". Modell ist also nicht eine Top-down-Struktur, in der um so mehr bestimmt je höher er(sie) sitzt, sondern eine vernetzte Struktur aus kleineren Einheiten. "Klein" ist dabei relativ und hängt von der Aufgabe ab.

3. Der Linux-Basar als Modell selbstgeplanter Wirtschaft

Nach so ausführlicher Vorbereitung liegen die Schlußfolgerungen nun auf der Hand. Keine neue Gesellschaft taucht aus dem Nichts auf und steht am nächsten Morgen vor der Tür. Keine neue Gesellschaft löst die alte ohne Widerstand ab. Zunächst entwickeln sich Keime des Neuem in den Nischen des Alten. Schließlich wird das Neue so mächtig, daß die Verwalter des alten Konzessionen machen müssen und das Neue gleichzeitig bekämpfen und verhindern wollen. Das Neue wird sich dann durchsetzen, wenn es effektiv besser ist als das Alte. Dabei ist es klüger, nicht auf dem ureigenen Terrain des Alten zu kämpfen, sondern die Spielregeln zu ändern und sich auf neuem Terrain zu behaupten.

Für solch ein Modell steht Linux und die freie bzw. quelloffene Software. Die Bewegung freier Software hat nicht einfach ein neue, bessere Firma gegründet und bessere proprietäre Software entwickelt (das hat Netscape versucht und ist gescheitert). Sie hat die Spielregeln verändert, den Quelltext offen gelegt und ein kollektives globales Entwicklungsmodell installiert. Interessant ist hierbei in langer Perspektive nicht das Produkt, sondern die neue Art und Weise der Produktivkraftentwicklung. Diskussionen über die Frage, ob freie Software eher zum Kapitalismus, zum Anarchismus oder zum Kommunismus kompatibel ist, gehen an der Sache vorbei [13]. Die Frage ist zunächst nicht, welche Gesellschaftsformation die angemessene ist, sondern wie die Arbeit beschaffen sein muß, damit sich in ihr der Mensch als Subjekt voll entfalten kann. Linux hat gezeigt,

daß das gehen kann.

Linux als Entwicklungsmodell nimmt einiges der neuen Gesellschaft vorweg. Wir beobachten, wie sowohl beim Einzelnen als auch im kollektiven Zusammenhang eines Projekts ein großer Kreativitäts- und Entwicklungsschub freigesetzt wird. Selbstentfaltung und Entfaltung der Anderen gehen hier schon tendentiell zusammen. Es gibt keinen großen Planer, der alle Projekte bestimmt, sondern jedes Projekt bestimmt sich selbst. Übergreifend gibt es sowohl Meetings und informelle Treffen wie auch die gemeinsame Vereinbarung verbindlicher Standards (z.B. als "Request for Commments": RFC in der offenen Internet Engineering Task Force: IETF, vgl. Bradner, 1999), an die sich die freien Gruppen eher halten als Softwarefirmen, die proprietäre Software entwickeln.

Wir beobachten auch die Tendenz der Kommerzialisierung der Bewegung freier Software. Das ist nicht verwunderlich, schließlich leben wir im Kapitalismus. Und es ist auch klar, daß große Firmen wie IBM, Netscape, Lotus etc. auf den fahrenden Zug aufspringen, um ihre Marktanteile - als Abstauber sozusagen - zu wahren. Diese Entwicklung muß man nüchtern beobachten und bewerten. Eine moralische Verdammnis individueller Handlungen ist unangebracht. Wer vom Kapitalismus nicht reden will, soll von der Verwerflichkeit der Kommerzialisierung der freien Software schweigen. Vielleicht ist es möglich, daß die freie Softwarebewegung über Patente und andere Restriktionen geknebelt wird. Vielleicht lassen sich auch Teile der Bewegung "einkaufen" und damit zurück in die proprietäre Software führen (z.B. bei Software mit "weichen" Nicht-GNU-Lizenzen). Aber die Idee und die Erfahrung der Power globaler, vernetzter und kollektiver Entwicklung von hervorragenden nützlichen Produkten für alle verschwindet nicht mehr.

Linux als Produkt *und* als neues Modell der Produktivkraftentwicklung ist "unabschaffbar"!

4. Meta-Text

4.1. Versionen-Geschichte

- Version 1.0: Grundlage für den Vortrag vom 24.06.1999 in Kaiserslautern
- Version 1.01: geringfügige Korrekturen, Ergänzung einer fehlenden Literaturangabe
- Version 1.02: Beseitigung eines kleinen semantischen Bugs
- Version 1.03: kleinere syntaktische und semantische Korrekturen
- Version 1.04: die NASA wurde erst nach der ARPA gegründet
- Version 1.05: Hinweis auf Papierversion eingefügt, Kapitel-Links gesetzt
- Version 1.06: Rechtschreibkorrekturen

4.2. Literatur

Bradner, S. (1999), The Internet Engineering Task Force, in: DiBona, C., Ockman, S., Stone, M. (1999), S. 47.

Brooks, F. P. (1995), The Mythical Man Month: Essays on Software Engineering, Reading/MA: Addison-Wesley.

DiBona, C., Ockman, S., Stone, M. (1999), Open Sources: Voices from the Open Source Revolution, Sebastopol/CA: O?Reilly; online verfügbar unter http://www.oreilly.com/catalog/opensources/book/toc.html.

Herrmann, J. (1983), Der Aufstieg der Menschheit zwischen Naturgeschichte und Weltgeschichte, Köln: Pahl-Rugenstein.

Marx, K. (1976), Das Kapital. Kritik der politischen Ökonomie. Erster Band, Frankfurt/Main: Verlag Marxistische Blätter. Identisch mit Marx-Engels-Werke, Band 23.

Meretz, S., Rudolph, I. (1994), Die >Krise< der Informatik als Ausdruck der >Krise< der Produktivkraftentwicklung, http://www.kritische-informatik.de/pk_inf.htm.

Meretz, S. (1999), Die doppelte algorithmische Revolution des Kapitalismus - oder: Von der Anarchie des Marktes zur selbstgeplanten Wirtschaft, http://www.kritische-informatik.de/algorev.htm.

Newman, N. (1999), The Origins and Future of Open Source Software, http://www.netaction.org/opensrc/future/oss-whole.html.

Perkins, G. (1998), Open Source and Capitalism, http://slashdot.org/articles/980824/0854256.shtml.

Raymond, E. S. (1997), Die Kathedrale und der Basar, Vortrag auf dem Linux-Kongreß Mai 1997, http://www.linux-magazin.de/ausgabe.1997.08/Basar/basar.html. Englischsprachige, überarbeitete Version: http://www.tuxedo.org/~esr/writings/cathedral-bazaar.

Raymond, E.S. (1999), How To Become A Hacker, http://www.tuxedo.org/~esr/faqs/hacker-howto.html.

Stallman, R. (1993), The GNU-Manifesto, http://www.gnu.org/gnu/manifesto.html; inoffizielle Übersetzung: Das GNU-Manifest, http://www.gnu.org/mani-ger.html.

Zakon, R.H. (1999), Hobbes? Internet Timeline v4.1, http://info.isoc.org/guest/zakon/Internet/History/HIT.html.

4.3. Anmerkungen

[1] GNU ist ein rekursives Akronym und heißt **G**NU Is **N**ot **U**NIX. Es drückt aus, daß das freie GNU-System funktional den proprietären Unix-Betriebssystemen entspricht, jedoch nicht wie diese proprietär, sondern frei ist. Zum Begriff "proprietär" siehe Anmerkung 3.

[2] Ein Programm, das im von Computer ausführbaren binären Format vorliegt, kann nicht geändert werden. Dazu ist der Quelltext (source code) des Programmes erforderlich.

[3] Proprietär heißt herstellerabhängig. Oft wird der Begriff zur Unterscheidung verwendet, ob Software einem "offenen Standard" entspricht oder nicht. Hier wird der Begriff eng verstanden: Jede Software, deren Quelltext der Hersteller nicht offen legt, ist von diesem abhängig. Folglich ist jede nicht-freie Software proprietär.

[4] Heute wird vielfach der Begriff des "hackens" mit dem elektronischen Einbruch in geschützte Computersysteme verbunden. Diese Fremd-Zuschreibung hat die eigentliche Bedeutung des Herstellens von innovativer Hard- und Software zersetzt. "Hacker" selbst bezeichnen diese Form des elektronischen Vandalismus als "cracken": "hackers build things, crackers break them." Vgl. Raymond, E.S. (1999).

[5] Interview des Online-Magazins Telepolis mit Richard Stallman: "Software muß frei sein!", http://www.heise.de/tp/deutsch/inhalt/te/2860/1.html

[6] Um freie Software-Bibliotheken auch in nicht-freier Software benutzen zu können, wurde die GNU Library GPL geschaffen, die diese Vermischung erlaubt (z.B. die GNU C-Library). Mit Version 2.1 wurde sie umbenannt in GNU Lesser GPL, vgl. http://www.gnu.org/copyleft/lesser.html

[7] Dokumentiert z.B. in DiBona, C., Ockman, S., Stone, M. (1999) im Anhang A oder im Internet unter http://www.lh.umu.se/~bjorn/mhonarc-files/obsolete/

[8] "Die Produktivkraft der Arbeit ist durch mannigfache Umstände bestimmt, unter anderen durch den Durchschnittsgrad des Geschickes der Arbeiter, die Entwicklungsstufe der Wissenschaft und ihrer technologischen Anwendbarkeit, die gesellschaftliche Kombination des Produktionsprozesses, den Umfang und die Wirkungsfähigkeit der Produktionsmittel, und durch Naturverhältnisse." (Marx, 1976, S. 54).

[9] So die Unterscheidung von Sklavenhaltergesellschaften und Patriachalischen Ausbeutergesellschaften, vgl. Herrmann (1983). Auch in den Übergängen zwischen den Formationen gab in der frühen Zeit beträchtliche Unterschiede. So vollzog sich im germanischen Raum der Übergang zur Feudalgesellschaft nicht über den Weg der Sklavenhaltergesellschaft.

[10] Aus Gründen der Vereinfachung ignoriere ich die Übergangsperiode vom Feudalismus zum Kapitalismus, die auch Manufakturperiode oder Frühkapitalismus genannt wird.

[11] Der Begriff geht auf Überlegungen der Kritischen Psychologin und Globalwissenschaftlerin Iris Rudolph (http://mitglied.tripod.de/IrisRudolph) zurück. In einem gemeinsamen Artikel haben wir ihn erstmals verwendet, vgl. Meretz, Rudolph (1994).

[12] Vielen Dank an Annette Schlemm, die mir hier auf die Sprünge geholfen hat. Vgl. http://www.thur.de/philo.

[13] Vgl. Perkins (1998) und die sich daran anschließende Debatte.

Quelle: http://www.kritische-informatik.de/linuxswl.htm
Ich bin an Deiner Meinung interessiert, schreib' mir: stefan.meretz@kritische-informatik.de