POSITIONSPAPIER

Freie Software/ Open Source

Hinweise für öffentliche IT-Dienstleister und ihre Kunden

Dr. Michael Neubauer, Citkomm
Stand: September 2011

VITAKO
Bundes-Arbeitsgemeinschaft der Kommunalen IT-Dienstleister e.V.

Inhalt

Zusammenfassung

Open Source-Software oder - wie es eigentlich korrekt heißen müsste - „freie Software“ ist seit rund 10 Jahren ein viel diskutiertes Thema in der öffentlichen Verwaltung. Dabei stehen häufig spektakuläre Entscheidungen im Vordergrund, die einen kommerziellen Softwarehersteller zu Gunsten einer freien Software vollständig ersetzen wollen und das auch medienwirksam inszenieren. Doch diese schwarz-weiß-Diskussion wird der Komplexität des Themas nicht gerecht. Freie Software ist zwar lizenzkostenfrei, die Einführung und der Betrieb erfordern aber erhebliche intellektuelle und organisatorische Kapazitäten, die in einer Gesamtrechnung mit berücksichtigt werden müssen.

Der Einsatz freier Software wurde primär aus Gründen der Lizenzkosten betrachtet. Tatsächlich ist er in erster Linie aus strategischen Gründen interessant. Freie Software bietet eine langfristige und nachhaltige Softwarenutzung mit einer hohen Flexibilität.

Das vorliegende Papier betrachtet kurz die wichtigsten Punkte, die beim Einsatz von freier Software zu berücksichtigen sind und macht konkrete Vorschläge, wann und wie freie Software eingesetzt werden kann.

Freie Software eignet sich besonders für den Einsatz in einem Rechenzentrum. Hier gibt es eine professionelle Betriebsmannschaft. Zusätzlich ist bei freier Software ein kleines Entwicklerteam für Automatisierungsaufgaben notwendig. Im Bereich der Serversoftware gibt es große freie Softwareprojekte, die über eine langjährige Kontinuität verfügen und so ein hohes Maß an Investitionssicherheit bieten. Die hohen Anforderungen an das Personal und die tendenziell längeren Implementierungszeiten führen aber auch dazu, dass kleinere Organisationen einen starken Dienstleister benötigen. Sonst können viele Vorteile von freier Software wieder verloren gehen.

Freie Software ist nicht sicherer als unfreie Software. Auch die Beseitigung von Fehlern ist nicht generell besser. Bei einer sorgfältigen Auswahl von leistungsfähigen freien Softwarekomponenten kann aber feststellt werden, dass Transparenz und Schnelligkeit bei der Fehlerkorrektur ein wesentlicher Vorteil gegenüber unfreier Software ist. Das gilt besonders für den Bereich der Serversoftware.

Bei Service und Wartung gibt es kein einheitliches Bild, da dieser Bereich sowohl bei freier Software als auch bei kommerzieller Software sehr differenziert zu betrachten ist - es gibt jeweils positive und negative Beispiele. Das gilt umso mehr, wenn der Kunde bereit ist, Geld für Service auszugeben. Denn gerade im Bereich der freien Software gibt es viele kleine und sehr kompetente Dienstleister, die den Vergleich mit großen Organisationen nicht zu scheuen brauchen.

Die Anwendung von freier Software auf dem Rechner des Anwenders hat sich trotz vieler Versuche nicht wirklich durchgesetzt. Die Gründe dafür sind vielfältig, im Ergebnis aber eindeutig: Während im Serverbereich kleinere Probleme durch eigene Entwicklungen und kompetente Mitarbeiter ausge-

glichen werden können, führen diese beim Endkunden zu dauerhafter Unzufriedenheit. Daher gibt es sicher viele benutzernahe Anwendungsbereiche, die mit freier Software abgedeckt werden können. Sie müssen aber genau analysiert werden und werden nur selten zu einer vollständigen Ablösung von kommerzieller Software führen können.

Im Server- und Grundlagenbereich ist freie Software heute für einen kommunalen IT-Dienstleister eine echte Alternative zu kommerzieller Software. Dabei geht es nicht um ein „entweder-oder", sondern um eine Abwägung der jeweiligen Anforderungen und um eine gemischte Nutzung. Der Einsatz muss langfristig geplant und nachhaltig umgesetzt werden, da mit der Nutzung von freier Software auch das entsprechende Personal verfügbar sein muss.

Für kleinere Organisationen mit wenig IT-Fachpersonal ist freie Software nur in Ausnahmefällen zu empfehlen, da die Einsparungen bei den Lizenzkosten schnell durch Dienstleistungskosten kompensiert werden können. Das Gleiche gilt für den vollständigen Einsatz am Arbeitsplatzrechner.

1 Einleitung

Freie Software ist heute ein fester Bestandteil in der kommunalen Softwarelandschaft. Die Einführung solcher Produkte ist aber nicht ohne Probleme verlaufen. Das vorliegende Papier fasst wesentliche Erkenntnisse und Empfehlungen zusammen, die sich aus der langjährigen Befassung des Autors mit diesem Thema ergeben. Darüber hinaus sind Erfahrungen aus dem Kreis der Mitglieder von Vitako, der Bundes-Arbeitsgemeinschaft Kommunaler IT-Dienstleister eingeflossen.

Der vorliegende Text gibt einige Hinweise und Erklärungen, die bei der Einführung und Nutzung von freier Software berücksichtigt werden sollten. Dabei geht es primär darum, kommunale IT-Dienstleister und deren Kunden bei der Entscheidungsfindung zu unterstützen. Das Thema ist komplex und in vielen Bereichen auch sehr heterogen. Aus diesem Grund ist dieses Papier keine Einführung und kein Ratgeber. Es soll vielmehr dazu dienen, eine Orientierung für eine strategische Ausrichtung zu finden. Aus diesem Grund richtet sich das Papier an die

- Geschäftsleitungen der IT-Dienstleister, die für ihr Unternehmen und mit den Kunden strategische Entscheidungen treffen und fundiert begründen wollen, und die
- Entscheiderebene der Verwaltung und den Kommunalpolitiker, die sich speziell mit der Steuerung der IT auseinander setzen müssen.

Wie häufig in der Informationstechnik ist die Nomenklatur auch bei „freier Software" uneinheitlich. Daher werden zunächst einige Begriffe genauer gefasst, um so eine gemeinsame Basis für das Verständnis des eigentlichen Textes zu schaffen.

Freie Software gibt es für fast alle Bereiche der Informationstechnik. Doch nicht überall hat sie die gleiche Qualität und Verbreitung. Im Kapitel über die Leistungsstrukturen (Kap. 4) wird eine grobe Kategorisierung des freien Softwaremarktes vorgenommen. Anschließend wird der Nutzen der verschiedenen Produktkategorien für den kommunalen Bereich eingeschätzt.

Der Siegeszug des Internets ist eng mit dem Erfolg von freier Software verbunden. Viele freie Softwareprodukte sind die Basis für normgebende Protokolle im Internet. Darüber hinaus sind die meisten Entwickler davon überzeugt, dass ihre Software dann genutzt wird, wenn sie mit vielen anderen Programmen gut zusammen arbeitet. Im Kapitel „Interoperabilität" (Kap. 5) werden diese Aspekte aufgegriffen.

Das Thema Sicherheit wird seit Jahren intensiv diskutiert. Lange Zeit galt freie Software per se als sicher. Das hat sich in den letzten Jahren gewandelt. Angesichts der hohen Komplexität und Emotionalität kann dieser Punkt hier nur angedeutet werden. Dabei geht es primär darum, die prinzipiellen Unterschiede zwischen den verschiedenen Entwicklungskonzepten zu erläutern.

Im letzen Kapitel über Investitionssicherheit werden die Vor- und Nachteile des Einsatzes von freier Software in Bezug auf die langfristig anfallenden Kosten dargestellt. Dieser Punkt wurde lange vernachlässigt, weil in vielen Investitionsrechnungen primär die Anschaffungs- und Einführungskosten betrachtet werden. Tatsächlich entstehen aber wesentliche Kosten in der Betriebsphase. Auch hier können keine generellen Aussagen getroffen werden. Es geht vielmehr darum, die verschiedenen Ansätze in ihren Grundstrukturen zu vergleichen.

Das Thema Kosten war für viele Anwender immer das zentrale Argument für die Verwendung freier Software. Dieser Aspekt wird hier nur am Rande im Kapitel über „*Investitionssicherheit und Nachhaltigkeit*" behandelt. Der ein oder andere Leser mag verwundert sein, dass genau der Punkt so wenig Raum in diesem Papier einnimmt, der doch das Hauptargument für die Anwendung von freier Software zu sein scheint. Die Fokussierung auf das Thema der Lizenzkosten fußt im Grunde auf einem Irrglauben. Erstens machen die Lizenzkosten an den Gesamtkosten nur einen relativ geringen Anteil aus, zweitens erfordert der strategische Einsatz von freier Software zusätzliche Aufwände bei der Evaluierung und Integration von Software, die zumindest im Allgemeinen die Kostenvorteile beim Lizenzkauf kompensieren.

2 Begriffe

Im allgemeinen Sprachgebrauch wird der Begriff Open Source-Software zum Teil missverständlich verwendet. Bezeichnet wird damit meistens quelloffene und lizenzkostenfreie Software. Bei einer genaueren Betrachtung des Begriffs zeigt sich jedoch, dass dies in Teilen irreführend ist. Da-

her wird in den folgenden Abschnitten eine weitergehende Begriffsbildung vorgenommen.(Stallman, 2010)

Im Folgenden wird von „offener“ Software gesprochen, wenn die Software quelloffen zur Verfügung steht. Hiermit ist aber nicht automatisch das Recht auf eine Veränderung oder eine Weitergabe der Software verbunden. Beispiele für offene Software, die weder frei zugänglich noch frei zu verändern sind, gibt es viele. So sind z.B. große Teile der Anwendungssoftware von SAP in der Programmiersprache ABAP entwickelt. Diese Software wird dem Kunden als Quellcode zur Verfügung gestellt.

Neben der Eigenschaft, dass eine Software „offen“ ist, stellt sich die Frage nach der Lizenzgebühr. Wir sprechen von einer kostenfreien Software, wenn der Lizenzgeber für diese Software kein Entgelt verlangt. Viele private Autoren stellen ihre Software kostenfrei zur Verfügung. Das geht aber nicht notwendigerweise auch mit einer Offenlegung des Quellcodes einher. Viele Autoren und Firmen veröffentlichen ihre Software zunächst kostenfrei, um so eine schnelle Verbreitung sicherzustellen. Nach einem gewissen Markterfolg sind sie dann immer noch frei, Lizenzgebühren für ihre Software zu verlangen.

Gerade bei Open Source-Software ist es wichtig, zwischen „kostenfreier“ und „freier“ Software zu unterscheiden. Freie Software gibt dem Lizenznehmer das Recht, die Software frei zu nutzen, ihre Funktionsweise zu verstehen, sie weiter zu verbreiten und zu verbessern

Umgangssprachlich wird Open Source häufig synonym mit freier Software verwendet. Da das begrifflich nicht korrekt ist, wird im Folgenden nur der genauere Begriff der „freien Software“ verwendet. Der Begriff Open Source wird nur für quelloffene Software verwendet, ohne damit ein besonderes Lizenzmodell festzulegen. Mit Closed Source oder „kommerzieller Software“ werden Lizenzmodelle bezeichnet, die eine Veränderung des Quellcodes verbieten.

3 Lizenzmodelle

Viele Akteure im Umfeld der freien Software vergleichen Gedankengut und Programme miteinander. Hieraus leiten sie die zunächst einmal frappierende Konsequenz ab, dass die Gedanken „frei“ sind und daher auch die aufgeschriebenen Programme nichts anderes sind, als formalisierte Gedanken. Die meisten Rechtssysteme gehen aber geradezu vom Gegenteil aus: Programme gehören dem Ersteller. Ihre Nutzung ist nur im Rahmen von Nutzungsrechten zulässig. Vor diesem Hintergrund ist es gar nicht einfach, Lizenzmodelle so zu gestalten, dass die Nutzung der Programme frei ist und frei bleibt.‚Frei‘ bedeutet, dass jeder die Software nutzen und verändern darf. Das schließt ausdrücklich nicht aus, dass mit der Nutzung Geld verdient wird.

Der Erfolg freier Software liegt ganz wesentlich in dem zugrunde liegenden Lizenzmodell. Hinter diesem Konzept stehen komplizierte, juristische und

strategische Überlegungen. (Stallman, et al., 2009) In diesem Papier werden nur einige besonders wichtige Aspekte angesprochen, die heute für das Verständnis und die Bedeutung von freier Software in der IT-Branche von zentraler Bedeutung sind.

Die Grundidee des Lizenzmodells freier Software ist, dass der Softwareentwickler dem Anwender weitreichende Nutzungs- und Änderungsrechte einräumt. Gleichzeitig behält der Entwickler alle Urheberrechte[1] an der Software und bietet durch einen Vertrag, der der Software beiliegt, folgende Lizenz an:

- die Software zu nutzen
- die Software zu verändern
- die Software weiterzugeben
- bei der Weitergabe der Software die Lizenzrechte einzuhalten.

Das Grundkonzept dieses Ansatzes hat die Entwicklergruppe GNU (namentlich Richard Stallman) entwickelt. Sie hat bereits vor vielen Jahren die sog. GNU-Public-Licence (GPL) entworfen. Viele Software-Autoren beziehen sich auf diese Lizenz und veröffentlichen ihre Software zusammen mit diesem Lizenzmodell. Inzwischen haben sich aber darüber hinaus viele weitere Lizenzformen etabliert, die zum Teil Mischformen von freier und unfreier Software sind.

Mit der Nutzung einer Software akzeptiert der Verwender z.B. bei der oben angesprochenen GPL die Lizenzbedingungen. Er verpflichtet sich gleichzeitig, Veränderungen unter dem gleichen Lizenzmodell bereitzustellen. Hiermit wird auch er gezwungen, die Veränderungen an der Software allen Verwendern zur Verfügung zu stellen. So wird verhindert, dass einmal als frei deklarierte Software durch Veränderungs- und Veredelungsprozesse zu unfreier und kostenpflichtiger Software wird. Daneben gibt es aber auch viele andere Lizenzmodelle, die weniger restriktiv sind. In jedem Fall sollte vor der Verwendung von freier Software eine intensive Auseinandersetzung mit den Lizenzbedingungen erfolgen.

4 Leistungsstrukturen freier Software

Freie Software gibt es heute für praktisch jeden Anwendungsbereich. Doch werden nicht alle Anwendungsbereiche durch freie Software in gleicher Weise unterstützt. Im Folgenden geben wir einen kleinen Überblick über die wichtigsten Kriterien, die bei der Auswahl von freier Software zu beachten sind und wo freie Software besonders gut eingesetzt werden kann.

[1] Im deutschen Rechtssystem sind Urheberrechte ohnehin unveräußerlich, das ist aber z.B. im angelsächsischen Bereich nicht so.

4.1 Marktstruktur

Viele freie Softwareprojekte nutzen für die Verwaltung die Internetplattform Sourceforge.[2] Dort sind zurzeit mehr als 100.000 Projekte mit einer GPL-Lizenz registriert. Rund ein Fünftel davon sind im Zustand „produktiv" (Sourceforge)[3] Das ist auf den ersten Blick ein erstaunlich hoher Wert. Doch eine genaue Betrachtung zeigt, dass sehr viele Projekte nur selten neue Versionen herausgeben und auch nur wenige Nutzer haben. Per se kein Nachteil, aber häufig doch ein Indiz dafür, dass das Projekt nicht gut unterstützt ist. Das kann Auswirkungen auf den Service und auch auf die Beseitigung von Fehlern haben (vgl. unten).

Jeder Leser kennt vermutlich Projekte wie den Apache-Webserver oder den Internetbrowser Firefox. Diese Projekte sind gut organisiert und verfügen zum Teil sogar über hauptamtliche Entwickler. Hier ist ein Einsatz ebenso unkritisch wie bei einem kommerziellen Produkt. In allen anderen Fällen sollten die in Kapitel 7 „*Investitionssicherheit und Nachhaltigkeit*" genannten Punkte genau bedacht werden.

Es gibt viele Spielarten von freien Softwareprojekten, doch lassen sich einige Grundstrukturen aufzeigen, die gerade unter dem Aspekt Nachhaltigkeit wichtig sind:

- **Freie Teams und Einzelentwickler:** Lange Zeit wurde freie Software fast ausschließlich von freien Entwicklern produziert. Sie arbeiten an der Software in der Freizeit und sind häufig selbst intensiver Nutzer der Software (Bazaarmodell, 2011).
- **Institutionelle, non profit Projekte:** Größere Projekte haben in aller Regel eine institutionelle Basis. So wird die grafische Benutzeroberfläche KDE von einem eingetragenen Verein getragen. Der Web-Browser Firefox wird von der Mozilla Foundation bzw. von deren Tochtergesellschaft entwickelt. Solche Institutionen finanzieren sich z.B. durch Spenden oder den Verkauf von Merchandising-Produkten. Eine solche institutionelle Basis ist zum Einen Garant dafür, dass das Produkt professionell entwickelt wird. Zum Anderen, dass die Ideale einer freien Softwareentwicklung gelten.
- **Kommerzielles Basisprodukt:** Heute ist es immer schwieriger, neue Softwareprodukte am Markt zu etablieren. Viele Unternehmen stellen ihre Produkte daher als freie Software zur Verfügung und verlangen keine Lizenzkosten, weil sie sich Service- und Wartungseinnahmen er-

[2] Sourceforge ist nur eine von vielen Plattformen, über die Entwickler freie Software verteilen können.

[3] Auf Sourceforge stehen umfangreiche Such- und Selektionsmöglichkeiten zur Verfügung. Der Qualitätszustand im Projekt wird von den Projektteilnehmern selbst verwaltet. Die Aussage, dass eine Software z.B. produktiv nutzbar ist, ist daher eine Selbsteinschätzung des Projekts.

hoffen. Hierbei kommen sowohl spezifische Lizenzmodelle der Hersteller als auch solche Modelle zum Einsatz, die den Prinzipien freier Software entsprechen.

Häufig wird auch ein sog. „Dual-Lizenzkonzept“ verwendet. Beispiele sind die Datenbank MySQL, OpenOffice oder das Virtualisierungssystem VirtualBox.[4] Die Software wird ganz oder teilweise unter GPL veröffentlicht. Gleichzeitig gibt es eine kommerzielle Software mit erweitertem Service oder zusätzlichen Funktionen. Bei diesem Modell muss die Herstellerfirma darauf achten, dass sie Verbesserungen aus dem Anwenderkreis nur in den kommerziellen Softwareteil übernimmt, für den die Rechte abgetreten wurden.

Viele Akteure sehen eine institutionelle Non Profit-Basis als Schlüssel für eine nachhaltige Entwicklung freier Software. Das schließt nicht aus, dass mit der *Nutzung* der Software Geld verdient wird. Vielfach ist das von den Entwicklern ausdrücklich gewünscht.

Die Ursprünge für eine institutionelle Non Profit-Basis kann aus beiden „Richtungen“ kommen. Einerseits gibt es viele Projekte, die sich erst mit der Zeit aus der Entwickler- und Nutzergemeinschaft, der sog. Community, zu einer Stiftung oder einem Verein formieren, andererseits gibt es Fälle wie Mozilla, wo eine Firma, in diesem Fall AOL, die Basis für einen institutionellen Rahmen schufen.

4.2 Anwendungsbereiche

Das Angebot freier Software, ist heute kaum zu überschauen. Aus Sicht der kommunalen IT-Dienstleister lassen sich aber grob unterscheiden:

- **Entwicklungswerkzeuge:** Große Teile der freien Software sind zunächst mit dem Ziel entwickelt worden, den Softwareerstellungsprozess zu vereinfachen und die Kosten für den freien Softwareentwickler zu reduzieren. Daher gibt es eine Vielzahl von Editoren, Werkzeugen und Compilern, die ermöglichen, ohne großes eigenes Investment freie Software zu entwickeln.
- **Infrastrukturkomponenten:** Das sind Softwaresysteme, die von grundsätzlicher und infrastruktureller Bedeutung für das Leistungsangebot im Betrieb von IT sind. Typische Beispiele sind das Betriebssystem Linux oder die Entwicklungsbibliotheken des GNU-Projektes.

[4] Dieses Geschäftskonzept ist aus Sicht vieler Akteure in der Open Source-Szene kritisch zu sehen, weil der Einfluss des betreuenden Unternehmens sehr groß ist. Außerdem kann durch den Verkauf des Unternehmens die Nutzung der Software faktisch stark eingeschränkt werden. Wie real diese Befürchtung ist, lässt sich schon daran erkennen, dass alle im Text genannten Projekte mittlerweile an den ORACLE-Konzern verkauft wurden.

- **Dienste:** Das sind Anwendungen, die in modularer Form Funktionen für den Anwender zur Verfügung stellen. Typische Beispiele sind Webserver, Mailserver oder Verzeichnisdienste. Die Abgrenzung zwischen Infrastruktur- und Dienstkomponenten ist fließend.
- **Anwendungen:** Anwendungen lösen isolierte Problemstellungen des Endanwenders. Typische Beispiele sind das Fotobearbeitungsprogramm Gimp oder die Bürokommunikationssoftware Open Office.

Am Anfang der Open Source-Bewegung[5] stand der Gedanke, einen preiswerten Zugang zu Entwicklungswerkzeugen zu schaffen. Die hohen Kosten für Editoren, Compiler und andere Werkzeuge führten dazu, dass freie Entwickler nur begrenzte Möglichkeiten hatten, auf Entwicklungswerkzeuge zuzugreifen. Die Kosten für eine professionelle Entwicklungsumgebung waren hoch. Entsprechend gibt es eine große Anzahl von Compilern und Editoren, die sich speziell an den Softwareentwickler wenden. Gleichzeitig war schon in den ersten Tagen dieser Bewegung klar, dass auch die Werkzeuge mittelfristig eine Betriebssystemplattform benötigen. Die Entwicklung eines freien Betriebssystems kam aber über viele Jahre nicht voran.

Historisch gesehen bilden die **Dienste** den Kern der freien Software. Schon in den 80er Jahren wurden Dienste wie NFS (ein Dateidienst für Unix-ähnliche Betriebssysteme) oder BIND (ein Dienst zur Umwandlung von symbolischen Namen in IP-Adressen) entwickelt. Daneben gab es die Idee, ein eigenes, freies Betriebssystem zu entwickeln. Die Arbeiten zogen sich aber über viele Jahre hin, ohne dass eine produktive Lösung zur Verfügung stand. Erst mit der Entwicklung von Linux wurde der kritische Moment für den erfolgreichen Einsatz von freier Software überschritten.

So verwundert nicht, dass es heute grundsätzlich möglich ist, alle wesentlichen Funktionen im Backend eines Rechenzentrums auf der Basis von freien Softwarekomponenten abzuwickeln. Vitako-Mitglieder machen von dieser Möglichkeit in unterschiedlichem Umfang Gebrauch. Häuser mit einer besonders offensiven Strategie für freie Software realisieren mittlerweile nicht nur die Server, sondern auch die Router vollständig mit freier Software. Die eingesetzten Komponenten sind zum Großteil seit Jahrzehnten vorhanden und verfügen über eine hohe Stabilität. Für viele Bereiche gibt es auch unterschiedliche Implementierungsvarianten, die je nach Funktion und Aufgabenspektrum unterschiedlich skaliert werden können.

Völlig anders stellt sich die Situation bei den Anwendungen dar. Auch hier gibt es ein großes Softwareangebot. Typische Beispiele sind der Web-Browser Firefox oder der E-Mail Client Thunderbird. Zudem gibt es viele Editoren, mit denen Zeichnungen oder Bilder bearbeitet werden können. Den meisten dieser Softwaresysteme ist eines gemein: Sie lösen nicht ein

[5] Damit sind die Akteure der Erstellung und Verbreitung von **freier** Software gemeint.

konkretes Anwendungsproblem, sondern sind tendenziell Werkzeuge zur generischen Problemlösung.

Deswegen gilt die Regel „je spezifischer eine Problemstellung, umso schwerer wird es, eine passende freie Software zu finden.“ So gibt es z.B.. für die Verwaltung eines Kalenders zwar etliche freie Softwarelösungen. Sie konnten sich aber aus verschiedenen Gründen nicht gegen die marktführenden Produkte der Firma Microsoft durchsetzen.

Bei konkreten Fragestellungen wie etwa der Finanzbuchhaltung, zeigt sich schnell, dass es heute kein freies Softwareprojekt gibt, das auch nur annähernd an den Funktionsumfang und die Qualität moderner kommerzieller Software heranreichen kann.

Für spezifische kommunale Aufgaben, wie das Einwohnerwesen oder das Personenstandswesen, gibt es keinerlei freie Softwarealternativen. Das ist im Übrigen nicht auf den öffentlichen Bereich beschränkt, sondern gilt auch für Branchen wie etwa die Versicherungswirtschaft oder das Bankenwesen. Hier zeigen sich die Grenzen des Modells freier Softwareentwicklung deutlich.

4.3 Fazit

Auch wenn die Strukturierung auf den ersten Blick holzschnittartig erscheinen mag: Abschließend lässt sich feststellen, dass freie Software bei kommunalen IT-Dienstleistern im Wesentlichen für

- Entwicklung,
- Betrieb von Infrastrukturen,
- Basisdienste

einsetzbar ist. Mit freier Software lassen sich komplexe Infrastrukturen betreiben und komplexe Softwaresysteme entwickeln. Darüber hinaus können viele Basisdienste für das Internet bereitgestellt werden.

5 Interoperabilität

Freie Software gilt in weiten Teilen der Fachpresse als besonders interoperabel. Immer wieder stufen Autoren, besonders aus dem Bereich der öffentlichen Verwaltung, den Standardisierungsprozess von freier Software als effizient und leistungsfähig ein. Im Folgenden werden wir auf einige dieser Aussagen etwas detaillierter eingehen, um eine angemessene Einordnung zu ermöglichen.

Wir beziehen uns im Folgenden auf Standards und nicht auf Normen. Eine Norm ist in diesem Zusammenhang ein wohldefinierter Prozess zur Definition eines Standards. Ein Standard ist eine einheitliche und verbreitete Übereinkunft in einem vorgegebenen technischen Bereich. Wie es zu solch einer Übereinkunft kommt, ist dabei nicht festgelegt.

5.1 Netz- und Betriebssystemstandards

Die Entwicklung freier Software ist eng mit der Entwicklung des Betriebssystems Unix verbunden. Jahrzehntelang wurden Anwendungen entwickelt, die mittlerweile auf den unterschiedlichsten Unix-Plattformen zur Verfügung stehen. Grundsätzlich kann festgehalten werden, dass über Jahrzehnte hinweg wichtige Basiselemente des Unix-Betriebssystems zumindest an den Hochschulen kostenfrei und quelloffen zur Verfügung standen. Dieser Umstand führte dazu, dass viele Forschungs- und Entwicklungsgruppen an den Hochschulen Anwendungen, Dienste und Protokolle entwickelt haben. Viele dieser Entwicklungen wurden zumindest im wissenschaftlichen Umfeld ausgetauscht, schrittweise weiterentwickelt und an neue Anforderungen angepasst.

Interessant und wichtig ist, dass viele Probleme zunächst durch ein Programm oder einen Dienst gelöst wurden. Beispiele sind das Filetransfer Programm FTP oder die netzbasierte Terminalemulation Telnet. In beiden Fällen hat zunächst ein Entwicklerteam sowohl die Dienst- als auch die Anwendungsseite gemeinsam entwickelt. Das Protokoll, das zwischen den Komponenten abläuft, wurde nicht in einem mühsamen Standardisierungsprozess definiert, sondern anhand von pragmatischen Entwicklungsentscheidungen proprietär umgesetzt. Erst im weiteren Verlauf und mit dem zunehmenden Erfolg einer solchen Lösung entstand das Interesse, das entsprechende Kommunikationsprotokoll offenzulegen und zu normen. Der Normungsprozess bestand nur darin, das faktisch bereits existierende Protokoll nachträglich und unabhängig von der Implementierung zu beschreiben. Die genaue formale Spezifikation ließ sich im Wesentlichen nur durch die als Quellcode vorliegende freie Software definieren.

Dieses Vorgehen ist auf der einen Seite ausgesprochen effizient, da nur solche Protokolle beschrieben werden müssen, die tatsächlich eine gewisse Verbreitung im Internet gefunden haben. Auf der anderen Seite jedoch hochgradig gefährlich, weil unter Umständen in der tatsächlichen Implementierung vorhandene Schwachpunkte mehr oder weniger unkritisch in den Standard übernommen werden. Auf dem Weg von der Softwareentwicklung zum -Internetstandard gibt es zwar verschiedene Möglichkeiten, über eine Art Peer Review offensichtliche Mängel anzusprechen, die Praxis zeigt aber, dass dieses Vorgehen signifikante Schwächen hat.[6] Als klassisches Beispiel für die hier angedeuteten Probleme gilt der Dienst des Domain Name Service (DNS). Dieser Fall ging vor einiger Zeit durch die Presse, weil es ernstzunehmende Sicherheitsprobleme in dem unterliegenden

[6] Neu Konzepte, Informationen, aber auch Standards werden in sog. RFCs (Request for Comments) niedergelegt. Im Fall von Standards werden diese auch in wichtigen Fällen durch andere Normungseinrichtungen übernommen. Der dahinter stehende Prozess ist interessanterweise selbst wieder (nachträglich) als RFC formuliert worden.(Bradner, 1996).

DNS-Protokoll gab, die sich mehr oder weniger unmittelbar aus einer Implementierungsschwäche des normgebenden Programms BIND erklären lassen.

Vor diesem Hintergrund wird deutlich, dass das oft hochgelobte Standardisierungskonzept für das Internet sehr effektiv ist. Gleichzeitig machen die Überlegungen aber deutlich: Viele Standardisierungen im Bereich des Internets sind effektiv, weil sie sehr schnell zu nutzbaren Ergebnissen führen - aber nicht immer effizient, weil die anschließende Fortschreibung der entsprechenden Normen unnötig komplex und fehleranfällig ist. Er unterscheidet sich dem Grunde nach auch nicht so sehr von den Standardisierungsprozessen bei Closed Source-Software. Denn auch hier werden häufig zunächst über viele Jahre Standards über die Einführung von bestimmten Programmen gesetzt. Erst später, wenn auch Drittanbieter Interesse an der Integration von Leistungen haben, werden entsprechende Abläufe beschrieben und einer Normung zugeführt.[7] Bei einer solch kritischen Betrachtung sind die Unterschiede im Normungsprozess z.B. zwischen der Standardisierung des Dateiformates PDF oder Microsoft Word und der von vielen freien Softwareprojekten strukturell gering.

5.2 Technische Umsetzung

Während der Standardisierungsprozess durchaus kritisch bewertet werden muss, kann der freien Software in der technischen Umsetzung in weiten Teilen attestiert werden, dass sie eine hohe faktische Interoperabilität sichert. Den Entwicklern freier Software geht es in erster Linie darum, dass ihre Software auch tatsächlich genutzt wird. Fragen wie Investitionssicherheit, Betriebssicherheit oder Upgrade-Fähigkeit sind von besonderer Bedeutung.

Daher sind die meisten freien Softwareprojekte sehr darauf bedacht, die Interoperabilität zu Fremdprodukten sicherzustellen. Das ist tendenziell bei Closed Source-Anbietern nur dann der Fall, wenn es einen gewissen „Marktdruck“ gibt. Der Autor selbst hat die Erfahrung gemacht, dass qualifizierte Fehlermeldungen meistens sehr ernst genommen werden. Fehlerbereinigungen liegen sehr schnell vor. Zudem werden häufig auch solche Softwaremodule, die sich offensichtlich nicht normkonform verhalten, wegen der kurzen Entscheidungswege und der pragmatischen Einstellung vieler Projekte im Rahmen einer Ausnahmebehandlung berücksichtigt. Dies ist

[7] Das Beispiel MS-Word ist dafür ein gutes Beispiel: Über Jahre war kein Standard verfügbar. Das Format wurde z.T. mit neuen Programmversionen geändert. Erst mit der Umstellung auf XML hat Microsoft eine offizielle Normung vorangetrieben. Der Unterschied zu vielen freien Softwareprojekten ist daher nicht methodisch, sondern nur darin begründet, dass Microsoft keinen Einblick in den Quellcode zuließ. Die Analyse der Formate war dadurch etwas schwieriger als bei freier Software.

besonders dort erforderlich, wo große multinationale Hersteller bestimmte Standards nur unvollständig und fehlerhaft implementieren.

Als Betreiber kommunaler Anwendungen mit langjähriger Erfahrung im gemischten Betrieb von freier Software- und Closed Source-Software konstatiert der Autor ganz klar, dass der Betrieb einer freien Software-Infrastruktur aus dem Blickwinkel der Standardisierung wesentlich unproblematischer ist als dies mit Closed Source-Software der Fall ist.

Wegen der technischen Interoperabilität von freier Software glauben viele Beobachter fälschlicherweise, dass auch der Standardisierungsprozess dem der Privatwirtschaft überlegen ist. Das ist zumindest so allgemein nicht richtig.

5.3 Interoperabilität und proprietäre Standards

Die Interoperabilität freier Software hat dort ihre Grenzen, wo wesentliche Marktstandards als proprietäre und ggf. geheime Standards implementiert werden. Hier müssen die entsprechenden internen Standards von der Open Source Community mehr oder weniger analytisch erhoben werden. Dieser Prozess, der unter Technikern gern als „Reverse Engineering“ bezeichnet wird, kann sich jeweils nur auf die aktuell zur Verfügung stehenden Versionen stützen. Darüber hinaus lassen sich nicht immer alle Aspekte eines Standards durch die reine Beobachtung eines Systems valide abbilden.

Einige Closed Source-Anbieter erweitern oder verändern mit jeder neuen Version den entsprechenden proprietären Standard, um freien Softwareentwicklern die Bereitstellung von interoperablen Komponenten zu erschweren. Im Ergebnis sind interoperable Komponenten für die neuesten Versionen meist fehlerhaft oder gar nicht verfügbar. Typische Beispiele sind das freie Softwareprojekt WINE, das eine unter Linux ablauffähige Emulation der Windows-API[8] zur Verfügung stellt. Hier ist nur ein Teil der Software tatsächlich ablauffähig. Inkompatibilitäten ergeben sich dadurch, dass Entwickler das entsprechende Windows API nicht normenkonform nutzen.

Ein anderes Beispiel ist das über viele Jahre aus dem LAN-Manager der IBM heraus entwickelte Netzwerk-Filesystem Samba. Auch hier baut Microsoft mit jeder Windows-Version neue Eigenschaften in das System ein. Auch wenn Microsoft in den letzten Jahren die Samba Entwickler Community in gewisser Weise unterstützt hat, gibt es in der Praxis viele Versionsinkompatibilitäten.

Die mangelnde Interoperabilität kann, wie diese Beispiele zeigen. viele Gründe haben:

- Sie kann ihre Ursache in nicht offengelegten Spezifikationen haben.

[8] Ein API (Application Programming Interface) dient dem Entwickler als programmatische Schnittstelle zu einem Softwaresystem.

- Sie kann an der zögerlichen Umsetzung von offenen Standards liegen.
- Sie kann in der schlechten Implementierung der offenen Standards bei einzelnen Marktteilnehmern liegen.

Es muss jeweils im Einzelfall geprüft werden, welche Implementierung die jeweiligen Anforderungen am besten abdeckt.

5.4 Fazit

Die Interoperabilität freier Softwarekomponenten im Bereich von Netzwerk und Basisbetriebssystemen hat heute eine hohe Qualität. Der Betrieb ist in der Praxis einfach und problemlos. Der Standardisierungssprozess ist an vielen Stellen fragwürdig, wird aber durch die Erfolge der Praxis relativiert.

Überall dort, wo ein IT-Dienstleister freie Softwarekomponenten integriert, erfordert die Herstellung interoperabler Systeme eine hohe Kompetenz, die gerade in kleineren Betrieben nur schwer sicherzustellen ist. Ein Betrieb der neuesten Versionen von Closed Source-Software zusammen mit freier Software ist in vielen Fällen nicht oder nur sehr eingeschränkt möglich.

Vor diesem Hintergrund bedarf es besonders aus der Perspektive der Interoperabilität einer genauen Analyse, welche Vor- und Nachteile das jeweilige Lizenzmodell für den Nutzer bringt.

6 Sicherheit

Das Thema Sicherheit wird im Zusammenhang mit freier Software besonders emotional diskutiert. Immer wieder gibt es die Behauptung, freie Software sei per se sicherer als z.B. gekaufte Closed Source-Software. Diese generelle Aussage ist sicher falsch. Eine notwendigerweise differenzierte Darstellung ist nicht Ziel dieses Papiers. Im Folgenden werden lediglich einige grundsätzliche Zusammenhänge erläutert, die dem Leser ermöglichen, sich unter Berücksichtigung seiner eigenen Situation eine Meinung zu bilden.

Grundsätzlich gibt es einen ganz wesentlichen Unterschied zwischen Open- und Closed Source[9]:

> *Während ein Sicherheitsproblem bei einer freien Software bis in den Quellcode hinein verfolgt werden kann und damit die Fehlerursache transparent ist, bleibt es bei der Closed Source-Software bei einer rein phänomenologischen Betrachtung des Problems.*

[9] Hier geht es anders als in den vorangegangenen Kapiteln um die Frage, ob die jeweilige Software tatsächlich quelloffen ist. Das dahinter liegende Lizenzmodell ist, anders als bei der freien Software, beim Thema Sicherheit unerheblich.

Mit anderen Worten: Bei freier Software ist die Ursache transparent, bei Closed Source-Software nicht. Experten bewerten diesen Unterschied differenziert:

- Transparenz ist **positiv**, weil Fehler schon präventiv am Quellcode festgestellt werden können. Außerdem können kritische Umgebungsparameter[10] leichter eingeschätzt und für Testfälle genutzt werden. Es gibt Bereiche, z.B. in der Kryptografie, in denen auch die Offenlegung der Quellen die Sicherheit grundsätzlich nicht in Frage stellt.
- Transparenz ist **negativ**, weil Probleme im Quellcode sofort ersichtlich sind und gezielt ausgenutzt werden können.

Die unterschiedlichen Auffassungen der Experten rühren vermutlich daher, dass diese Vor- und Nachteile nicht symmetrisch für die Intransparenz gelten:

- Intransparenz ist **positiv**, weil Fehler nur indirekt durch das Verhalten einer Software erkannt werden. Daraus muss sich nicht notwendigerweise auch ein Sicherheitsproblem ergeben.
- Intransparenz ist **negativ**, weil nur der Hersteller Fehler beseitigen kann. Der muss dazu intellektuell in der Lage sein und über die notwendigen personellen Ressourcen verfügen.

Es soll und kann nicht entschieden werden, welche Position die **richtige** ist. Offensichtlich ist aber, dass die schnelle Beseitigung im Mittelpunkt steht, wenn Fehler nicht von Beginn an zu vermeiden sind. Doch gerade dies bereitet vielen Softwareherstellern Probleme.[11]

Offensichtlich ließe sich Vieles verbessern, wenn Sicherheitsprobleme gar nicht erst aufträten. Genau hier wird der Unterschied zwischen freier und offener Software wichtig. Denn selbst wenn die Software quelloffen ist, darf sie nicht unbedingt frei geändert werden. Freie Software bietet die Möglichkeit, dass ein Fehler unmittelbar nachdem er erkannt wurde, auch beseitigt wird. Bei vielen Projekten findet genau dies statt. Zum einen in der Phase der Softwareerstellung, in der die Akteure regelmäßig-- manchmal täglich- neue Versionen erzeugen und testen. Zum andern in der Produktion, wenn neue Fehler bekannt werden. Das ist keine Theorie, sondern täglich gelebte Praxis. Häufig gibt es sogar mehrere Instanzen, die sich mit der Fehlerbeseitigung beschäftigen:

- **Kunden** und Anwender, die eigene Entwicklungskapazitäten haben.
- **Distributoren** wie z.B. Red Hat, Suse oder Debian, die Fehler auf der Ebene ihres Quellcodestamms beseitigen.

---

[10] Beispiele sind ein schlecht konfiguriertes Berechtigungssystem oder fehlerhafte Systemeingaben.

[11] Die Zero Day Initiative (ZDI) moniert seit langer Zeit, dass viele Fehler von den Softwareherstellern nur sehr zögerlich beseitigt würden (*Vgl.* (ZDI, 2011)*)*

- **Entwickler** des Programms, die die Pflege und Weiterentwicklung des Projekts übernehmen.

Das alles führt dazu, dass bei einem gut unterstützten freien Softwareprodukt Sicherheitsprobleme in aller Regel schnell beseitigt werden. Das Qualitätsniveau ist häufig sehr hoch, teilweise besser als bei vielen Großunternehmen. Positiv ist auch, dass freie Softwareentwickler Fehler offen zugeben können, da sie keine Qualitätsversprechen gegeben haben. Gerade großen Firmen fällt ein Fehlereingeständnis oft sehr schwer. Entsprechend lang kann dann auch die Beseitigung dauern.

Kritisch sind solche Projekte, die nur wenige Entwickler betreuen oder der Nutzerkreis selbst wenig Unterstützung leistet. In diesem Fall gibt es oft keine Unterschiede zu den kommerziellen Angeboten. Im Gegenteil, während viele Unternehmen zumindest gegen eine finanzielle Beteiligung mit der Fehlerbeseitigung beginnen, ist das bei freien Entwicklern schwieriger. Zum einen gibt es Entwickler, die Geldzahlungen aus ethischen Gründen ablehnen, zum anderen haben sie ggf. keine Zeit oder Lust, sich mit diesen Problemen zu beschäftigen.

7 Investitionssicherheit und Nachhaltigkeit

Schon zu Beginn haben wir darauf hingewiesen, dass die Lizenzkosten bei der wirtschaftlichen Betrachtung einer Softwarelösung eine tendenziell geringe Bedeutung haben. Der Grund dafür ist einfach:

- Der Aufwand für Einführung und Betrieb der Software liegt um ein Vielfaches über dem der Lizenzkosten.
- Software wird viel länger genutzt als gewöhnlich angenommen. So haben z.B. nach einer Messebefragung der Fa. Mondula aus dem Jahr 2010 fast 50% der 217 Befragten erklärt, dass ihre Finanzsoftware seit mehr als 10 Jahren im Einsatz sei (Becker, et al., 2010).
- Die Integration neuer Software und die Anpassung neuer Softwareversionen ist ein Kostenbestandteil, der gerade bei der Einführung unterschätzt wird. Er entsteht aber potentiell über den gesamten Softwarelebenszyklus und wird durch die Upgrade-Zyklen vieler Anbieter von Closed Source-Software noch getrieben.

Wegen der genannten Phänomene **kann** freie Software große Vorteile haben.

7.1 Erfolgskriterien

Viele freie Softwareprojekte folgen einigen Grundregeln, die deren langfristige Nutzung und deren Kosten signifikant verringern. Diese Regeln stellt Gancarz (Gancarz, 1994) ausführlich dar und werden oft als die „10 goldenen Regeln der Unix-Entwicklung“ bezeichnet. Einige besonders wichtige Punkte sind:

- **Modularität**: Jedes Programm (Modul) soll nur eine Aufgabe erfüllen. Komplexe Aufgaben werden durch die programmatische Kombination einzelner Programme abgebildet.
- **Einfachheit**: Eng verbunden mit der Modularität ist die Forderung nach einfachen Lösungen. Dahinter steht die Erkenntnis, dass manche komfortable Lösung auf Dauer schwer zu warten ist.
- **Plattformunabhängigkeit**: Programme sollen so gestaltet sein, dass sie auf unterschiedlichen Systemen ablaufen können. Dabei ist die zu realisierende Funktion wichtiger als deren effiziente Implementierung.
- **Flexibilität**: Viele Anwendungen führen den Nutzer z.B. durch eine grafische Benutzeroberfläche, die eine schnelle Einarbeitung ermöglicht. Mittelfristig beschränkt aber ein solches „Korsett" die Nutzung, so dass der Benutzer in seiner Flexibilität eingeschränkt wird.

Genau genommen lassen sich diese Punkte natürlich auch auf Closed Source-Projekte sinnvoll anwenden. Doch die Interessen der großen Anbieter sind meistens andere, die zumindest auf den ersten Blick den Kundeninteressen sogar scheinbar eher entgegenkommen als das bei freier Software der Fall ist: Sie wollen dem Kunden möglichst viele Lösungen aus einer Hand anbieten. Ihre Lösungen lassen sich daher einfach installieren, bieten alle Funktionen und sind auf zertifizierten Plattformen getestet. Die Integration von Drittherstellern ist zwar meist grundsätzlich möglich. In diesem Fall steigen aber die Kosten und vor allem die Risiken, weil der Kunde die Verantwortung für das Zusammenspiel der Systemteile übernehmen muss.

Gute freie Software-Projekte sind überhaupt nur möglich, weil sie viele Basis-Module aus anderen Projekten nutzen. Eine Implementierung ganzer Software-Systemlandschaften, wie sie multinationale Unternehmen anbieten, würden deren personelle Ressourcen übersteigen. Die systemtechnische Integration ist entweder Aufgabe der Distributoren (z.B. SuSe, Red Hat) oder des Kunden selbst. Mit anderen Worten: Während im Closed Source-Bereich der Anbieter für die Integration zuständig ist, muss beim Einsatz freier Software der Kunde oder ein Dienstleister diese Funktion übernehmen.

Aus diesen Überlegungen lassen sich weitreichende Schlussfolgerungen ziehen:

- Der Einsatz freier Software erfordert eine leistungsfähige IT-Abteilung oder entsprechende externe Dienstleister, die in der Lage sind, Software selbst zu integrieren. Erforderlich ist sowohl Systemsoftware- als auch Entwicklungs-Knowhow.
- Die Installation und Projektierung von freien Softwareprojekten erfordert tendenziell eine aufwändigere Planung als bei Closed Source-Projekten.

- Freie Software lässt sich i.d.R. relativ unabhängig von Upgrade-Zyklen betreiben, wenn die Software modular aufgebaut ist.[12] Wegen des offenen Quellcodes und des Rechts auf freie Änderung der Quellen lassen sich auch alte Softwareversionen noch nach Jahren an die aktuellen Bedürfnisse des Betriebs anpassen.
- Viele freie Softwareentwickler haben ein großes Interesse daran, dass ihre Software breit genutzt wird. Aus diesem Grund achten sie besonders darauf, dass Schnittstellen und Standards genau eingehalten werden und langfristig stabil bleiben.
- Selbst beim Wechsel von Basiskomponenten, wie dem Betriebssystem oder des Applikationsservers, ist ein Weiterbetrieb möglich, weil die oben angesprochene Portabilität eine Migration ermöglicht.

Alle diese Aussagen haben eins gemeinsam: Sie gelten für viele Projekte, aber sie gelten nicht generell. Natürlich gibt es Ausnahmen. So hat z.B. das KDE-Projekt, das eine windowsähnliche Benutzeroberfläche erstellt, mit jeder neuen Version größere Änderungen an den Aufrufschnittstellen vorgenommen. Auch bei freier Software gibt es Sackgassen, die eine vollständige Neuorientierung erforderlich machen. Trotzdem gelten die Aussagen oben häufig. Denn selbst wenn eine neue Version vollständig inkompatibel zur alten ist und der Entwickler die alte Version nicht mehr unterstützt, gibt es immer wieder Nutzer, die die alte Version noch benötigen und daher weiter pflegen. Vielfach werden sogar Sicherheitsupdates aus der neuen in die alte Version übernommen (sog. Backports). Fazit: der Einsatz freier (modularer) Software garantiert - die entsprechende Kompetenz vorausgesetzt - eine hohe Unabhängigkeit und Investitionssicherheit.

7.2 Personal

Software ist nur mit gut ausgebildeten und motivierten Personal langfristig nutzbar. Die Frage nach der Investitionssicherheit ist daher nicht nur ein Frage der Softwarearchitektur oder deren Pflege. Es geht auch darum, dass genügend Personal für die verschiedenen Leistungsaspekte wie Kundenbetreuung, Störbeseitigung etc. zur Verfügung stehen.

Die Beschaffung von qualifiziertem IT-Personal wird aus verschiedenen Gründen in den kommenden Jahren schwierig:

- Der demografische Wandel wird generell zu einem Fachkräftemangel führen.

[12] Das gilt allerdings nur, wenn der Kunde die Modularität und Installation verstanden hat. Das ist z.B. nicht der Fall, wenn statt einer eigenen Installation auf eine sog. Appliance zurückgegriffen wird.

- Der Hochschulstudiengang Informatik wurde vor rund dreißig Jahren erstmals angeboten. In den kommenden Jahren werden die ersten Jahrgänge in den Ruhestand kommen.

Wir kämpfen zukünftig mit zwei gegenläufigen Trends, die sich verstärken. Einerseits werden weniger Informatiker ausgebildet, anderseits werden mehr Fachkräfte aus der IT in den Ruhestand gehen.

Diese Effekte haben zunächst keinen unmittelbaren Bezug zu den hier behandelten Themen. Zusammen mit einem dritten Trend entsteht aber ein grundsätzliches Problem für viele Closed Source-Anbieter und –Anwender: An den Hochschulen wird aus vielen Gründen primär mit freier Software entwickelt. Hier liegen die Quellcodes offen. Hier können ohne Probleme Erweiterungen und Verbesserungen vorgenommen werden.

Die so ausgebildeten Informatiker werden auch primär solche Arbeitgeber suchen, die sich mit ähnlichen Techniken wie die Hochschule beschäftigen. Hier rächt sich z.T. auch eine Strategie vieler in den USA beheimateten Softwareanbieter, ihre Entwicklungsabteilungen sehr zentral aufzustellen. Hoch qualifizierte Informatiker finden im Bereich der Closed Source-Software in Europa nur Arbeit beim Anwender. Doch das wird von vielen Hochschulabgängern nicht als Herausforderung gesehen.

Vor diesem Hintergrund ist die Zukunft der freien Software auch im Bereich Personal als grundsätzlich positiv einzustufen.

8 Zukunft von freier Source

Die Frage nach der Zukunft von freier Software ist nur schwer zu beantworten. Grundsätzlich kann festgestellt werden, dass es heute ein solch breites Angebot gibt, dass freie Software auf viele Jahre mit Sicherheit einen festen Platz in der „Softwarelandschaft“ haben wird. Ob sich der in den letzten Jahren festzustellende Wachstumstrend so fortsetzen wird, hängt von vielen Punkten ab:

- Viele erfolgreiche freie Softwareprojekte werden z.T. massiv von IT-Unternehmen unterstützt. Diese Firmen ziehen aus diesem Engagement im Gegenzug strategische und finanzielle Vorteile. Wie lange diese Symbiose von freier Software und kommerziellen Interessen trägt, ist unklar. Hier gilt es besonders die Projekte zu beobachten, die im eigenen Haus eingesetzt werden.
- Viele freie Softwareprodukte entstehen, weil Entwickler Lösungen für ein persönliches Problem benötigen. Dieser Bedarf wird ganz wesentlich auch davon getrieben, dass es viele individuelle Softwaresysteme und –anwendungen gibt. Neuere Entwicklungen, wie z.B. die des Cloud Computing, könnten zu einer Verringerung der IT-Diversität führen. Damit könnte es auch zu einem Rückgang der freien Softwareprojekte kommen.

Im Grunde gilt bei freier Software genauso wie bei jeder anderen Software auch, vor dem Einsatz zu prüfen, ob diese neben den fachlichen Anforderungen auch langfristig Bestand haben kann. Dabei sollte u.a. genau geprüft werden, wie viele Entwickler die Software entwickeln und wie die Community aussieht, die das Produkt unterstützt.

Literaturverzeichnis

Bazaarmodell (2011) Wikipedia - Bazaarmodell. *www.wikipedia.de.* [Online] 1. 3 2011. http://en.wikipedia.org/wiki/The_Cathedral_and_the_Bazaar

Becker, Marco und Ortmann, Jan (2010) *ERP-Studie.* Hamburg : mondula GmbH, 2010.

Bradner, S. (1996) The Internet Standards Process -- Revision 3. *IETF Tools.* [Online] 10 1996. [Zitat vom: 08. 03 2011.]

Gancarz, Mike (1994) *The UNIX Philosophy.* s.l. : Digital Press, 1994.

Sourceforge.*www.sourceforge.net.* [Online] [Zitat vom: 1. 3 2011.] www.sourceforge.net/search.

Stallman, Richard M. (2010) Why Open Source misses the point of Free Software. *GNU Operating System.* [Online] 1. 10 2010. [Zitat vom: 8. 03 2011.] http://www.gnu.org/philosophy/open-source-misses-the-point.html

Stallman, Richard M., Gay, Joshua und Lessig, Lawrence (2009) *Free Software Free Society: Selected Essays of Richard M. Stallman, 2nd Edition.* s.l. : Createspace, 2009.

ZD (2011) ZDI benennt Sicherheitslücken bei Microsoft, IBM, HP, Novell. *www.heise.de.* [Online] Heise Verlag, 8. Februar 2011. [Zitat vom: 28. 2 2011.] http://www.heise.de/newsticker/meldung/ZDI-benennt-Sicherheitsluecken-bei-Microsoft-IBM-HP-Novell-1185355.html

Copyright: Vitako – Bundes-Arbeitsgemeinschaft der Kommunalen IT-Dienstleister

Markgrafenstraße 22 – 10117 Berlin – Tel 030-2063156-0 – info@vitako.de