Freie Universität Berlin

Seminar „Ausgewählte Beiträge zum Software-Engineering“

Sommersemester 2008

Haftungsproblematik bei quelloffenen Software-Projekten (FOSSP)

István Bartkowiak
istvan.bartkowiak@inf.fu-berlin.de
Betreuer: Martin Gruhn

16. Oktober 2008

Zusammenfassung

In FOSS-Projekten erfolgt in weltweiter kooperativer Zusammenarbeit die Entwicklung von Software, deren Quellen für jedermann zugänglich sind. Die Verwendung derartiger Software kann jedoch Probleme aufwerfen, die über einfache Mängel hinausgehen. Erleidet der Benutzer Schäden oder sieht sich Ansprüchen Dritter ausgesetzt, stellt sich sogleich die Frage der Haftung. Diese Arbeit beschreibt aus dem Blickwinkel eines Softwareentwicklers Situationen und Probleme, die in der Praxis auftreten können. Dabei werden sowohl lizenzrechtliche als auch vertragsrechtliche Aspekte im Kontext der deutschen Rechtsordnung beleuchtet. Den Abschluss bildet die Vorstellung zweier bekannter FOSS-Projekte und deren praktisch erprobte Arbeitsweise.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Bei einer ersten Betrachtung der Haftungsproblematik stellt man schnell fest, dass eine scheinbar unüberschaubare Vielfalt von Problemen im praktischen Umfeld existiert. Allen Problemen gemein ist jedoch, dass sie sich in einzelne Fallgruppen unterteilen lassen. Dieses Prinzip des Teilens und Herrschens ist gemeinhin bekannt. Wendet man es in diesem Kontext an, so erkennt man schnell, dass die Existenz einer vertraglichen Bindung relevant sein muss. Entsprechend spricht man von vertragsrechtlichen Verstößen.

Auf den zweiten Blick offenbart sich jedoch eine zweite Gruppe, die gänzlich unabhängig berücksichtigt werden muss. So ist es durchaus möglich, sich Ansprüchen Dritter ausgesetzt zu sehen, mit denen man vorher in keinerlei Beziehung stand. Ein Kontakt kommt erst durch den Verstoß zustande, der Rechte jenes Dritten verletzt, der diese nun durchzusetzen sucht. Diese Fallgruppe beschreibt urheberrechtliche Verstöße.

Wenn es vorkommen kann, dass Haftungsfragen auch bei quelloffener Software relevant sind, steht die Frage nach der Motivation im Raum. Es müssen Argumente existieren, die die Risiken kompensieren oder sogar einen Mehrwert begründen. Eben solche Argumente sorgten in der jüngeren Vergangenheit für den Erfolg von FOSS im kommerziellen Umfeld. Einerseits verspricht der kollektive Ansatz stark verkürzte Produktzyklen, andererseits eine drastische Senkung der Entwicklungskosten durch den unbeschränkten Zugriff auf einen globalen Erfahrungsschatz. Damit einhergehend ist eine Besserung der Flexibilität festzustellen, die auf der leichten Anpassbarkeit derartiger Software an eine bestehende Infrakstruktur gründet. Und schließlich stehen auch handfeste finanzielle Erwägungen auf der Habenseite, wenn es darum geht, Nutzungsrechte zu erwerben. Eine essentielle Eigenschaft ist die unentgeltliche Lizenzierung [1].

Dennoch können diese Vorteile die grundsätzlich ablehnende Haltung bei Neuerungen nicht vollends entkräften. So wird gerne ins Feld geführt, dass es bei der Integration von FOSS als ergänzender Bestandteil zu Problemen mit bestehender CSS (Closed Source Software) kommen kann. Insbesondere bei einem breit angelegten Umstieg ist es unabdingbar, zu jeder Zeit den Überblick zu behalten. Die gemeinnützig tätige und herstellerunabhängige OSA (Open Solutions Allliance) hat diesbezüglich eine interessante Lösung entwickelt. Da es problemlos möglich ist, die Steuerung und Rückmeldungen von FOSS auf Grundlage des Quelltextes zu analysieren, kann auch die Einbindung in eine standardisierte Verwaltungsumgebung erfolgen, dem *Common Customer View*. Dabei handelt es sich um eine einheitliche Oberfläche für den Betrieb und die Steuerung unterschiedlichster FOSS. Der Ansatz ist jung und muss sich erst in der Praxis beweisen, zeigt jedoch das hohe Maß an Bereitschaft, Probleme nachhaltig durch saubere Konzeptionierung zu lösen anstatt sich an einen oder wenige Hersteller zu binden, um ähnliches zu erreichen. Dennoch ist diese Heterogenität symptomatisch und vielschichtig. Es hilft, sie auf einer höheren Abstraktionsebene zu betrachten. Durch den enorm anwachsenden Quelltextbestand bei gemeinschaftlich entwickelter Software steigt ebenso die Anzahl der beteiligten Entwickler an, gänzlich unabhängig davon, ob diese nun als angestellte Entwickler oder extern Beteiligte tätig sind. Eine Bestandsaufnahme im Sinne einer Inventur oder auch nur die endgültige und eindeutige Feststellung der Urheberschaft einzelner Quelltextteile wird zunehmend erschwert, letztlich sogar unmöglich.

Daraus abzuleiten, dass spätestens beim Punkt Verantwortung und Zuweisung von Aufgaben FOSS unbeherrschbar wird, ist voreilig und vielfach sogar falsch. Interessant sowohl für Dienstleister als letztlich auch für den Anwender sind stets die Reaktionszeiten, in denen entdeckte Mängel behoben werden. Der durch den Gesetzgeber z. B. in Deutschland eingeräumte

zeitliche Handlungsspielraum bei der Mängelbehebung hilft in der Praxis kaum weiter, geht es doch bei sicherheitskritischen Mängeln eher um Stunden als um Tage oder gar Wochen. FOSS zeichnet sich in diesem Bereich, eine vergleichbare Anzahl von Entwicklern vorausgesetzt, durch eine deutlich höhere Agilität aus. Betrachtet man etwa Projekte wie *Apache* (siehe auch Kapitel 3.1 auf Seite 22), deren störungsarmer Betrieb des dort entwickelten *HTTP Server* zu dessen Erfolg beigetragen hat, so kann eine wohldefinierte Vorgehensweise bei der Fehlererkennung, -berichterstattung und -behebung festgestellt werden. Je besser die Organisation im Projekt abläuft, desto effektiver lassen sich die Reaktionszeiten im Ernstfall verkürzen. Kleine Projekte haben hier einen entscheidenden Nachteil, da die erwartete Reaktionszeit mit der Anzahl der aktiven Entwickler korrelliert.

Nicht minder wichtig ist die Kontinuität eines FOSS-Projektes. Es mag für freie Entwickler attraktiv sein, sich in bekannte Projekte einzubringen, um sich dort eine gewisse Reputation zu erarbeiten, aber auch Ein-Mann-Projekte können erstaunliche Innovationen und qualitativ hervorragende Softwareprodukte hervorbringen. Dennoch ist deren Fortbestand unmittelbar von deren Ziehvater abhängig. Stellt dieser seine Arbeiten mangels Interesse oder Änderung der Lebensumstände ein, wird damit meist auch das Projekt beendet.

Besondere Bedeutung beim Einsatz von FOSS im professionellen Einsatz kommt der ausgeübten Mangelbehebung im Sinne des Einspielens einer Korrektur zu (Patch Management). Durch den bereits erwähnten heterogenen Charakter ist die IT-Abteilung zumeist mit einer beachtlichen Anzahl verschiedener Softwareprodukte konfrontiert, die jeweils auf dem aktuellen Stand gehalten werden müssen. Leider gibt es in dieser Hinsicht bisher noch keine allgemein anwendbare Automatik, um dem Problem zuverlässig zu begegnen. Im professionellen Umfeld sind es meist mittlere bis große Unternehmen, die über genügend fachliche Kompentenz verfügen, spezifische Automatismen zu entwerfen und zu implementieren. Dort ist es ein zentrales Erfordernis, über genaue Kenntnisse bezüglich der verwendeten Software und deren Versionsstand zu verfügen. Ein Mischbetrieb verschiedener Softwaregattungen verschärft dieses Problem weiter. Es ist daher durchaus normal, dass Änderungen vielerorts manuell eingespielt werden müssen. Besonders bei hochverfügbaren Systemen ein mühseliges Unterfangen, gilt es doch die Korrekturen vorher auf deren Verträglichkeit zu überprüfen.

Sowohl in Umgebungen, in denen CSS und FOSS zugleich Anwendung finden als auch bei reinen FOSS-Systemen kann es zu weiteren Risiken kommen, die sich direkt aus den verwendeten Lizenzen und den extern bezogengen Teilen ergeben:

1. **Inkompatible und virulente Lizenzen** — Findet z. B. Copyleft-basierte Software (siehe auch Kapitel 2.1 auf Seite 7) Anwendung, so ist das Unternehmen bei der Entwicklung und Lizenzierung eigener Softwareprodukte nicht unerheblich eingeschränkt. Gleiches gilt für die vorgelagerte Eigennutzung fremder Software, deren Lizenzen dem Lizenznehmer Pflichten auferlegen, die andere Lizenzen verbieten. Als Beispiel sei hier die Pflicht zur exklusiven Lizenzierung von Weiterentwicklungen an vorher festgelegte Kunden genannt.

2. **Stillgelegte Systemteile** — Moderne Softwareprodukte weisen insbesondere im professionellen Umfeld eine nahezu unüberschaubare Funktionsvielfalt auf, die das Ergebnis einer langen Entwicklungsgeschichte ist. Einzelne Teile können dabei stillgelegt worden sein, weil andere deren Funktion übernommen haben oder sie absichtlich durch lizenzvertragliche Vereinbarungen deaktiviert wurden. Kommt es durch eine fehlerhafte Konfiguration oder Mängel im weitesten Sinne dennoch zu deren Ausführung, sind Probleme bei der Implementation oder Integration zu erwarten.

3. **Unentdeckte Komponenten von Drittherstellern** — Ebenso wie inaktive Bereiche kann es Systemteile geben, die Komponenten von Drittherstellern enthalten, ohne dass dies zum Zeitpunkt der Inbetriebnahme bekannt war. Existiert für eben diese Komponenten keine Lizenzvereinbarung, so kann dies äußerst umfangreiche Umbaumaßnahmen nach sich ziehen oder der Zwang zum Erwerb einer Lizenz bestehen.

4. **Geerbte Lizenzverletzung** — Setzt sich ein System aus mehreren, teilweise separat lizenzierten Teilen zusammen, kann es zur Fortführung von Lizenzverstößen kommen. Dies ist in allen Fällen denkbar, bei denen Software von Zulieferern oder externen Projekten stammt.

5. **Kombination** — Jedes der obigen Risiken birgt enorme Schadenspotentiale. In der Praxis kann es dennoch zu einer zusätzlichen Verstärkung kommen. So ist ein paralleles Auftreten durchaus denkbar.

Untersucht man konkrete Beispiele im FOSS-Bereich, so wird die Gegenwärtigkeit von Lizenzproblemen sofort ersichtlich. Allein das Projekt *Debian* besteht aus über 20.000 Paketen, die in der Summe über 300.000 Lizenzhinweise enthalten [2]. Es liegt daher nahe, zumindest in dieser Hinsicht durch entsprechende Werkzeuge, die gebotene Übersicht wiederherzustellen. Eine mögliche Lösung liefert das Projekt *FOSSology*, welches sich der automatischen Analyse von Lizenzhinweisen und -texten widmet [3]. Das verfolgte Ziel dabei ist die Widerspruchsfreiheit bei Copyleft-Lizenzpflichten sicherzustellen und Konfliktbereiche zugänglich zu machen. In diesem Kontext hat sich auch ein spezieller Begriff für die Abänderung oder Rekombination bekannter OSD[1]-konformer Lizenzen durchgesetzt: *Frankenstein Licenses*. Eine genaue Abgrenzung von Pflichten und Verträglichkeiten der Lizenzen zueinander kann bei dieser Art der „Adaption“ unmöglich werden.

Da *FOSSology* selbst ein FOSS-Projekt ist, steht es in direkter Konkurrenz zu bestehenden kommerziellen Lösungen. Entsprechend wird es vom SFLC (Software Freedom Law Center) mittlerweile sogar explizit empfohlen, um unbeabsichtigten Verstößen, z. B. bei der Verwendung der GPL (General Public Licence), vorzubeugen [4]. Für die Anwendung in der Praxis wird die Vorgehensweise besprochen, *FOSSology* als zentralen Lizenzmanager einzusetzen, um eine Harmonisierung der unternehmens- und projektinternen heterogenen Lizenzlandschaft zu erreichen. Dabei kann das Werkzeug mittlerweile mit über 300 unterschiedlichen Lizenzen umgehen und den lizenzbezogenen Istzustand eines Projektes übersichtlich präsentieren.

Damit die im Rahmen einer OSD-konformen Lizenz gewährten Nutzungsrechte wahrgenommen werden können, muss der Lizenznehmer gewisse Pflichten beachten. Nur so gelangt er in den Genuss der begehrten Rechte. Dabei handelt es sich im Vergleich zu proprietären Lizenzen meist um eher kleine Pflichten. So besteht in der Regel lediglich die Pflicht zur namentlichen Nennung des Urhebers, einem Hinweis auf die Art der Lizenz zusammen mit einem geeigneten Urheberrechtshinweis und der Auflistung vorgenommener Modifikationen. Ganz klar verboten ist das Entfernen derartig bereits vorhandener Hinweise, z. B. der Umwidmung der Urheberrschaft auf den eigenen Namen. Bei Verstößen ist es wichtig, zwischen Vertrag und Lizenz zu unterscheiden. So kann ein Projekt durchaus im Auftrag tätig sein und auf Grundlage eines entsprechenden Werkvertrages eine Software entwickeln, welche im Anschluss sogleich unter einer OSD-konformen Lizenz der Öffentlichkeit zugänglich gemacht wird. Hat diese Software nicht oder nur teilweise die vertraglich vereinbarten Eigenschaften,

[1] OSD — Open Source Definition, s. `http://de.wikipedia.org/wiki/Open_Source_Definition`

liegt eine Verletzung des Vertrages vor. Verstößt dagegen ein Benutzer der Software gegen Pflichten im Rahmen der Lizenz, so liegt eine Verletzung der Urheberrechte vor. Beides wird im Detail in Kapitel 2.2 auf Seite 16 bzw. Kapitel 2.1 auf Seite 7 besprochen. Die Haftung im Kontext von FOSS-Projekten ist somit grundsätzlich nicht anders gelagert als bei proprietär entwickelter Software. Vielmehr stellt sie sich systemimmanent und rückgreifend auf alle involvierten Teile dar. Lizenzverstöße für sich genommen, können durchaus Haftungsansprüche begründen. Insbesondere dann, wenn FOSS in kommerziellen Produkten Anwendung findet und der Hersteller seinen Pflichten nicht nachkommt. Wird das Produkt dennoch in den Verkehr gebracht, liegt unmittelbar ein Verstoß gegen die Verbreitungsrechte des Urhebers vor.

1.1 Fehlendes Problembewusstsein

Grundsätzlich sind OSD-konforme Lizenzen als solche wenig problematisch. Ihre Risiken gewinnen sie vielmehr aus ihrem Variantenreichtum. Bereits kleine Unterschiede sind das Ergebnis eines dogmatischen oder pragmatischen Unterscheidungsbebürfnisses. So kann es durchaus zu Missverständnissen bezüglich der tatsächlichen Risiken kommen. Wo eine Lizenz gerade Risiken sowohl für den Urheber als auch den letztlichen Verwender der Software aus dem Weg räumen soll, kommt es durch unüberlegtes Handeln schnell zu vermeidbaren Lizenzverstößen [2]. Im unternehmerischen Umfeld kann es durchaus passieren, dass sich niemand der Verantworlichen darüber im Klaren ist, dass FOSS im Betrieb Anwendung findet. Einer Studie von *Forrester Research* zufolge, wurde in 46 % der untersuchten technologieaffinen Unternehmen FOSS in den betrieblichen Abläufen entdeckt, ohne dass das jeweilige Unternehmen Kenntnis davon hatte [5]. Als Faustregel lässt sich sogar formulieren, dass der unternehmerische Quelltextbestand mindestens fünfmal mehr FOSS- und Drittherstellerkomponenten enthält, als vom betroffenen Unternehmen vermutet [1].

Mittlerweile ist auch im FOSS-Lager ein Sinneswandel eingekehrt, was den Umgang und die Notwendigkeit zur Lizenzierung der entwickelten Software anbelangt. Spätestens die Diskussion um das Reizthema Softwarepatente (siehe dazu auch Kapitel 2.1.2 auf Seite 11) hat prominenten FOSS-Projekten vor Augen geführt, wie selbstverständlich sich proprietäre Hersteller bei den Innovationen der Gemeinschaft bedienen und daraus ihren Gewinn generieren. Neben willentlich agierenden Unternehmen gibt es jedoch auch solche, die ohne Kenntnis etwaig involvierter OSD-konformer Lizenzen die Vermarktung ihrer Aufbauprodukte durchführen. Es ist daher geboten, die Unternehmen dahingehend zu sensibilisieren. So wird beispielsweise die Erstellung und Pflege eines Lizenzstammbaumes nahegelegt, der bei der späteren Vermarktung herangezogen werden kann. Auch hier macht ein Beispiel die Notwendigkeit unmittelbar deutlich. *Linux* in der Gestalt eines äußerst komplexen Makroprojektes verknüpft über 100 unterschiedliche Lizenzen im Sinne der OSD. Verfolgt man den Stammbaum weiter bis auf Paketebene, so setzt sich die Aufsplitterung in Einzellizenzen weiter fort. Es gibt inkludierende Pakete der Form „under GPL“, exkludierende der Form „not under GPL“, frei mit anderen OSD-konformen Lizenzen rekombinierende und sogar adaptierende Pakete, die das Wagnis der eingangs erwähnten Frankenstein-Lizenzen eingehen. Dass FOSS-Projekte mangels vertraglicher Verbindlichkeiten selten die Notwendigkeit einer Lizenzverwaltung sehen, mag noch nachvollziehbar erscheinen. Dass dies jedoch auch bei kommerziell agierenden Unternehmen der Fall ist, wirft ein besonderes Licht auf deren Professionalität. Dem Problem nachhaltig zu begegnen, beginnt bereits bei der Gründung eines neuen Projektes. So sollte die Art der Lizenzierung immer zentraler Punkt bei der Entscheidung sein, die Quellen eines fremden Projektes als Grundlage für das eigene zu verwenden. Im großen Maßstab haben sich hierzu

einzelne Unternehmen in der *FOSS-Bazaar Community* zusammengeschlossen, zu der Größen wie *HP*, *Coverty*, *Google*, *Linux Foundation*, *SourceForge* u. a. gehören.

Die Behandlung des Komplexes der Lizenzfragen setzt jedoch voraus, dass Quelltexte externen Ursprungs noch als solche identifizierbar und abgrenzbar sind. In der Praxis kommt es hier jedoch häufig zu Problemen. So ist es bei selbstorganisierenden FOSS-Projekten, die sich besonders freizügig bei der Einbringung von Quelltexten freiwilliger Entwickler zeigen, in der Vergangenheit häufig zur Kontamination des Quelltextbestandes durch Verschmischung und Durchsickern von Quelltexten zweifelhafter Herkunft gekommen. Insbesondere den kombinierten Einsatz von CSS und FOSS im unternehmerischen Umfeld begleitet diese latente Gefahr.

1.2 Wahrnehmung von Pflichten

Will man Lizenzverstößen wirksam begegnen, muss man nach den Ursachen suchen, die diese erst ermöglicht haben. Zum einen sind dies die einfache Nichtbeachtung oder ein grundlegendes Missverständnis über Art und Umfang einer konkreten OSD-konformen Lizenz. Vielfach kann auch beobachtet werden, dass Kommunikationsdefizite zwischen den Entwicklern und dem Management vorliegen. Die genannten Probleme sind allerdings durchaus behebbar [4]. Verfolgt man die Stationen beim Vertrieb von FOSS bevor sie beim Anwender ankommt, so kann man bereits Defizite beim Vertrieb feststellen. So ist den meisten Distributoren nicht bewusst, dass sie sich durch den Vertrieb von GPLv2-lizenzierter Software verpflichtet haben, die Quelltexte mindestens drei Jahre ab dem Zeitpunkt der Verbreitung der letzten Binärfassung vorzuhalten und auf Nachfrage bereitzustellen. Die jüngere GPLv3 verlängert diese Frist gar auf den letzten veröffentlichten Teil einer Software. Eine derart lange Frist kann dabei die branchenüblichen Produktzyklen bei weitem übersteigen und erfordert daher einen bestimmbaren Aufwand.

Vergegenwärtigt man sich die Bedingungen der GPL Version 2 (kurz: GPLv2) im Vergleich zur Version 3, so kann man auch bei den Lizenzen eine stete Entwicklung beobachten. Verpflichtete Version 2 den Lizenznehmer zur Vorhaltung der Quellen für drei Jahre auf physischen Datenträgern, so wertet Version 3 die öffentlich zugängliche Vorhaltung z. B. im Internet als hinreichend. Wichtig hierbei ist die Bereitstellung gegenüber jedem, der legitimer Empfänger der Binärfassung ist. Also auch gegenüber Dritten, nicht nur gegenüber den eigenen Kunden. Ein Delegieren dieser Vorhaltungspflicht ist im übrigen unzulässig. Ein kommerzieller Händler ist dazu verpflichtet, die Vorhaltung im Sinne der GPL sicherzustellen. Eine wirksame Lösung des Problems kann darin bestehen, keine reinen Binärfassungen anzubieten, sondern stets eine Kombination aus Quellen und Binärfassung.

1.3 Zuständigkeiten und Aufgabenverteilung

Neben den Fragen, die sich aus der Lizenzierung ergeben, stellen sich jene, die sich mit der eigentlichen Mängelbeseitigung am Softwareprodukt befassen. Große FOSS-Projekte betreiben zur Lösung dieser Frage ein öffentliches Mängelverzeichnis, in dem Anwender und Entwickler gleichermaßen über entdeckte Mängel berichten können. Diese Berichte werden meist einzeln ausgewertet (Selektion) und den zuständigen Entwicklern zugewiesen (Delegation). Einige Projekte verfahren auch nach dem Selbstbedienungsprinzip. Sobald über einen Mängel berichtet wird, steht es jedem der aktiven Entwickler frei, sich diesem anzunehmen. Grundsätzlich ist diese Vorgehensweise dennoch mit praktischen Problemen behaftet, da mit wachsender Pro-

jektgröße die Selektion nicht mehr handhabbar wird. Erschwerend kommt die ebenso wachsende Komplexität hinzu. Nicht jeder berichtete Mangel lässt sich einem Teil eindeutig zuordnen, so dass in einigen Fällen die Bearbeitung durch mehrere Entwickler notwendig ist. Auch hier wird ähnlich wie bei der Lizenzierungsproblematik nach einem Automatismus gesucht, der mit der Projektgröße Schritt hält. Dabei stellt sich unmittelbar die Delegation als nicht-triviales Problem heraus.

In einem Experiment wurden daher unterschiedliche Ansätze des maschinellen Lernens miteinander verglichen, um den bedingt kreativen Prozess der Mängeldelegation zu automatisieren [6]. Neben einer Stützvektormaschine (SVM) und einem stochastischen Klassifizierer kam auch ein Verfahren zum Einsatz, welches auf Entscheidungsbäumen beruht. Das Ziel des Experimentes sollte es sein, anhand von eintreffenden Berichten eine Klassifizierung vorzunehmen und anschließend eine möglichst kleine Gruppe von Entwicklern zu bestimmen, die den Mangel beheben können. In dem genannten Bericht wurde als Beispiel das FOSS-Projekt *Eclipse* herangezogen, für das Beobachtungsdaten über den Zeitraum 1. Januar 2005 bis 30. August 2005 vorlagen. Die Kosten für die Delegation der in diesem Zeitraum eingetroffenen Mängelberichte bezifferte man auf durchschnittlich zwei Personenstunden pro Tag – Arbeitskraft, die nicht mehr für die Entwicklungsarbeit zur Verfügung stand. Zwar waren die Ergebnisse eher ernüchternd, zeigten jedoch, dass die Automatisierbarkeit auch stark vom Projekt selbst abhing. So kann ein erster Automatismus immerhin die Möglichkeit eröffnen, den Delegationsprozess zu beschleunigen oder eine Vorsortierung vorzunehmen. Beides mit dem Ziel, die Entwickler in dieser Hinsicht zu entlasten [7].

Eine weitere Erkenntnis, die derartige Versuche zeigen, ist die Priorität der Mängelbehandlung. Da die Delegation ein sehr sensibeler und fehlerträchtiger Vorgang ist, können falsche Zuordnungen gerade bei sicherheitskritischen Mängeln zu gravierenden zeitlichen Verzögerungen führen. Wie komplex der Entscheidungsvorgang ist, zeigt sich dahingehend, welches domänenspezifische Wissen zur Entscheidungsfindung herangezogen wird. So muss der Entscheider nicht nur genaue Kenntnis über das Projektziel besitzen, sondern auch über die Personalstruktur innerhalb des Projektes. Des weiteren sind Kenntnisse über die Individualkompetenzen der Entwickler und die zeitlichen Rahmenbedingungen im Sinne der Erfüllung von Meilensteinzielen maßgeblich. Dem Rechnung tragend weist beispielsweise das Verwaltungssystem *Bugzilla* den Beteiligten einzelne Rollen zu, um die Arbeit effizient zu verteilen. Es gibt Berichterstatter, die neue Berichte einreichen und Sortierer, die diese prüfen und die kritische Delegation vornehmen. Die Instandsetzer wiederum, beheben das ihnen zugewiesene Problem und können auf die Arbeit von Unterstützern zurückgreifen. Das Verwaltungssystem liefert zu jedem Zeitpunkt einen Überblick über den Stand der Arbeiten und erlaubt bei besonders kritischen Sachverhalten ein zielgerichtetes und koordiniertes Vorgehen.

2 Problemstellungen

Der Ansatzpunkt für Haftungsfragen ist in der Verwertung einer Software als Produkt zu suchen. Also Probleme, die sich beim Wahrnehmen von Nutzungs- und Verwertungsrechten ergeben. So stellt die Verwendung einer proprietär entwickleten Software bereits eine Verwertung dar, da für die Ausführung der Software eine entsprechende Lizenz vom Hersteller benötigt wird. Darüber hinausgehende Verwertungen gestalten sich wegen des Fehlens des Quelltextes schwierig und könnten höchstens darin bestehen, die Binärfassung der Software zu vervielfältigen und zu vertreiben. Im Gegensatz dazu ist bei FOSS eine Vielzahl von

Verwertungsmöglichkeiten denkbar. So bildet sie regelmäßig die Grundlage für nachgelagerte Dienstleistungen, mit denen Gewinne erwirtschaftet werden. Derartige Dienstleistungen können zum Beispiel der Vertrieb in Kombination mit zusätzlichem Dokumentationsmaterial sein, aber auch die Anpassung der Software an Kundenwünsche und die Betreuung bei Betrieb und Wartung. Die treibende Kraft bei der Verwendung von FOSS ist die Unabhängigkeit vom eigentlichen Hersteller der Software. Dienstleistungen können von beliebigen Zulieferern bezogen werden, wodurch eine hohe Flexibilität bei der Auftragsvergabe erreicht wird. Eben diese Flexibilität kann mittel- bis langfristig als Garant für eine Herstellerunabhängigkeit ihre volle Wirkung entfalten.

Neben den wirtschaftlichen und planungsstrategischen Erwägungen spielen weitere Faktoren eine wesentliche Rolle. So kann einerseits das *Geheimnisprinzip* als Grundpfeiler angeführt werden. Sowohl im unternehmerischen Umfeld, bei behördlichen Instanzen als auch beim Endverbraucher spielt die Geheimhaltung sensibler Daten eine enorme Rolle. Insbesondere Berufsgeheimnisträger im Sinne § 203 StGB, also z. B. Anwälte, Ärzte, Steuerberater oder Amtsträger, müssen sicherstellen, dass sie ihrer Verschwiegenheitspflicht hinreichend nachkommen. Dies gelingt allerdings nur, wenn die Art und Weise der Verarbeitung, Speicherung und Übermittlung sensibler Daten nachvollziehbar und belegbar ist. Dieses Prinzip ist auch als Kerckhoffs'sches Prinzip[2] aus der Kryptographie bekannt, wonach die Sicherheit eines Verfahrens nicht von dessen Geheimhaltung abhängig sein darf, sondern lediglich von den Eingabegrößen. Um dieser Anforderung gerecht zu werden, bedarf es der Offenlegung der Quelltexte. Nur so können sicherheitsrelevante Schwachstellen zuverlässig aufgedeckt werden [8].

Den zweiten Grundpfeiler bildet die *Kontrollierbarkeit des Informationsaustausches*, z. B. bei behördlichen Instanzen. So untersagt die deutsche Rechtsordnung beispielsweise die Kenntnisweitergabe zwischen Nachrichtendiensten und Polizeibehörden, da erstere über weiterreichende Kompetenzen und Möglichkeiten verfügen als bei der polizeilichen Ermittlungstätigkeit zulässig sind. Dieses Prinzip wird auch als Trennungsgebot bezeichnet.

Auch aus Herstellersicht kann die Freigabe der Quelltexte durchaus lohnenswert sein. Einer der Verlustfaktoren ist die Mängelbehebung im Rahmen der gesetzlichen Gewährleistungspflicht. Diese Pflichten sind mit Kosten verbunden, die den Gewinn beachtlich schmälern können. Wird die Mängelbehebung dagegen an die Nutzergemeinde delegiert, so kann sich der Hersteller der Software nahezu vollständig von diesen gesetzlichen Pflichten befreien. Im Vergleich zum konventionellen Vertrieb bringt dies Vorteile und verschafft dem Unternehmen einen legalen Vorteil gegenüber seinen Konkurrenten. Des weiteren ist auch ein Mischvertrieb denkbar wie dies der Softwarehersteller *Sun* sehr erfolgreich mit seiner Bürosoftware vollführt. So übernimmt die Mängelsuche und -behebung weitgehend die Anwender- und Entwicklergemeinde des FOSS-Projektes *OpenOffice.org*. Von dieser Software wird in regelmäßigen Abständen das proprietär lizenzierte Pendant *StarOffice* abgeleitet, welches durch Erweiterungen für den Kunden attraktiver erscheinen soll.

2.1 Lizenzrechtliche Aspekte

Zunächst muss beachtet werden, dass die Quelloffenheit keineswegs von einer konkreten Lizenz oder Lizenzart abhängt. So sind die Quelltexte auch von einigen proprietären Softwareprodukten zugänglich, meist allerdings mit der wesentlichen Einschränkung, dass Einsicht nur bestimmte Personenkreise erhalten. Ebenso geschieht die Offenlegung in der Regel nicht

[2] `http://de.wikipedia.org/wiki/Kerckhoffs_Maxime`

freiwillig, sondern um den Anforderungen von Ausschreibungen oder gesetzlichen Auflagen zu entsprechen. Schließlich stellt für einen proprietären Hersteller das Softwareprodukt einen zentralen Wert dar, der augenblicklich verloren ginge, wenn die Quelltexte veröffentlicht würden. Vielmehr ist das Vermarktungsmodell insbesondere bei Standardsoftware auf den entgeltlichen Verkauf von Nutzungslizenzen ausgerichtet und basiert auf der Geheimhaltung der Quellen und deren Einordnung als Unternehmensgeheimnis.

Die eigangs erwähnte *Open Source Definition* beschreibt im Sinne einer Rahmenlizenz grundlegende Eigenschaften einer Lizenz, um als OSD-konform zu gelten. Diese Mindestanforderungen definieren die Quelloffenheit nur als eine Eigenschaft. Des weiteren muss die Weiterverbreitung und Veränderung der Quellen unentgeltlich durch den Urheber eingeräumt werden. Die noch weiter greifende Eigenschaft des sogenannten *Copyleft*[3] ist dabei keine notwendige Anforderung, wenngleich jedoch wünschenswert. So ist beispielsweise die LGPL durchaus OSD-konform, obwohl sie die Weiterverwertung der Änderungen nicht einschränkt. Diese können auch proprietär vertrieben werden, wobei die LGPL-lizenzierten Teile auch dann FOSS bleiben.

Betrachtet man Software als geistiges Werk im erweiterten Sinne, so ergibt sich dennoch ein Unterschied zum klassischen Verständnis. Software existiert sowohl als Quelltext mit Kommentaren und Annotationen als auch als maschinenlesbare Repräsentation, der sogenannten Binärfassung. Es ist daher hilfreich, Software in ihrer Gesamtheit, als Entität, mit dem Begriff des geistigen Werkes zu verknüpfen. Des weiteren muss berücksichtigt werden, dass Software durchaus den Urheber eindeutig identifizieren kann, also „für ihn steht“. Ist das der Fall, kann eine Veränderung der Quelltexte zu einer Entstellung des Werkes führen, von der sich der Urheber distanzieren möchte oder diese sogar untersagt [8]. Diese Problematik tritt analog sehr häufig bei den *Lizenzen für freie Inhalte* auf, die in vergleichbarer Weise zu den OSD-konformen Lizenzen die öffentliche Zugänglichkeit sicherstellen.

Für die korrekte praktische Anwendung und Sicherstellung der GPL-Konformität hat das SFLC einen Praxisleitfaden veröffentlicht, der FOSS-Projekte bei der Beachtung der Regelungen unterstützen soll [3]. So kam es in der Vergangenheit häufig auch durch Fehlinterpretationen zu Verstößen – sowohl bei den Projekten selber als auch bei den Distributoren. Das von dem Leitfaden verfolgte Ziel soll dabei sein, ein besseres Verständnis der GPL im Kontext praxisnaher Fragestellungen zu vermitteln und Hinweise für die optimale Abstimmung zwischen Entwicklern und Management zu liefern. So wird beispielsweise ausdrücklich die eingangs erwähnte Lösung empfohlen, die Quelltexte stets gebündelt mit der Binärfassung auszuliefern, so dass alle Distributionswege implizit abgedeckt werden. Ferner ist es zulässig, die Unkosten für die Erstellung und Zusendung von etwaig notwendigen physischen Datenträgern dem Empfänger der Quellen in Rechnung zu stellen. Freilich unter der Vorgabe, stets eine möglichst günstige Herstellungsmethode zu wählen [4]. Des weiteren wird ausdrücklich darauf hingewiesen, dass es zu einer automatischen Beendigung des Lizenzvertrages kommt, wenn die Pflichten der GPL nicht beachtet werden. Die automatische Wiederherstellung, also eine nachträgliche Herbeiführung der Wirksamkeit, ist in der deutschen Rechtsordnung jedoch als problematisch einzustufen. Die im Rahmen eines GPL-basierten Softwarelizenzvertrages gewährten Nutzungsrechte können vom Lizenznehmer nur dann in zulässiger Weise wahrgenommen werden, wenn dieser den damit einhergehenden Pflichten nachkommt. Da ein Nichtbeachten zum sofortigen Verlust dieser Rechte führt, bedeutet eine Fortführung der weitergehenden Nutzung, die über die reine Ausführung der Software hinausgeht, eine Ur-

[3] `http://de.wikipedia.org/wiki/Copyleft`

heberrechtsverletzung. Gegen diese kann der Urheber, also stets der Autor der Quelltexte, vorgehen.

Eine saubere Quelltext- und Revisionsverwaltung wird auch von Seiten des SFLC empfohlen. Alle im Rahmen des Projektes generierten Daten sollten zentral verwaltet werden, so dass ein automatischer Aufbau des Lizenzstammbaumes möglich wird und Probleme frühzeitig und zuverlässig erkannt werden können. Insbesondere kleine Projekte leiden unter der Symptomatik, dass es häufig zu einer Kompetenzzentralisierung kommt. Projektentwickler, die den Status eines sogenannten *Build Gurus* erlangt haben, sind insoweit problematisch, dass deren Beteiligung am Projekt unabdingbar ist. Verlässt ein derartiger Entwickler das Projekt, ist der Verlust nur schwer kompensierbar. Um diesem Problem vorbeugend zu begegnen, sollten alle Entwickler mit dem Prozess der Versionserstellung vertraut sein. Dies kann nur dann erreicht werden, wenn die Methodik im Projekt wohldefiniert ist. Standardisierung und Wiederverwendung von bereits erprobten Verfahren und Prozessen im Umfeld von FOSS-Projekten sollten bei neuen Projekten vorzuziehen sein. Eine derartige Herangehensweise wird auch als *FOSS-Governance* bezeichnet [2].

Betrachtet man die GPL genauer, so offenbart dies Details, die überraschend erscheinen können. So wird gemäß Ziffer 6 GPLv2 bzw. Ziffer 10 GPLv3 stets der Autor der Quelltexte als lizenzgebende Person bestimmt. Eine Übertragung der Lizenzierungsrechte an einen Händler oder Distributor ist somit nicht möglich [8]. Als zweites interessantes Detail lässt sich anführen, dass die Erhebung eines beliebigen Entgeltes für eine dem Erwerber der Software gewährte Garantie gemäß Ziffer 1 Abs. 2 GPLv2 bzw. Ziffer 4 Abs. 2 GPLv3 problemlos zulässig ist. Die GPL als populärste Lizenz hat in diesem Zusammenhang vornehmlich regulierende Funktion. Es gilt, die Einräumung und Beschränkung von Rechten des Lizenznehmers zu definieren, so dass sich daran weiterführende vertrags- und haftungsrechtliche Bestimmungen knüpfen lassen. In diesem Sinne fehlt der GPL jedoch eine Steuerungsfunktion, da sie dem einfachen Verwenden der Software keine Schranken auferlegt. So darf man GPL-lizenzierte Software problemlos auf den eigenen Systemen ausführen – eine Pflicht zur Ausführung oder weitergehenden Verwendung besteht dagegen nicht. Vielmehr zeigt sich die Motivation für die Lizenzierung unter der GPL facettenreich.

1. **Dogmatische Überzeugung** — Eine unkomplizierte Anwendung und Bewährung in der Praxis soll auch andere zur Verwendung bewegen, um somit die Gemeinschaft nachhaltig und stabil zu vergrößern.

2. **Pragmatische Überzeugung** — Diese Art der Lizenzierung ist ein erprobter und etablierter Weg, Software zu veröffentlichen und dennoch die Kontrolle zu behalten, da proprietärer Missbrauch explizit von der Lizenz genannt und als unzulässig deklariert wird.

3. **Erzwungene Überzeugung** — Durch das Prinzip des Copyleft erlangt die GPL virulenten Charakter und zwingt Aufbauprojekte, selbst unter dieser Lizenz ihre Software zur Verfügung zu stellen.

Als problematisch erweist sich in diesem Zusammenhang die Mehrfachlizenzierung. Gemäß Ziffer 5 Satz 2 GPLv2 bzw. Ziffer 9 Satz 3 GPLv3 ist deren Zulässigkeit ungeklärt. Die dortigen Formulierungen beziehen sich ausschließlich auf den Lizenznehmer. Entsprechend besteht für den Urheber keine Verpflichtung, ausschließlich die GPL zu verwenden. Die Anwendung weiterer Lizenzen wirkt durch den ergänzenden Charakter jedoch nicht auf nachgelagerte Be-

arbeitungen der Quellen durch Lizenznahme auf Grundlage der GPL. Die Urheber von Weiterentwicklungen sind nicht verpflichtet, ebenfalls die Mehrfachlizenzierung vorzunehmen.

2.1.1 Urheberrechte

Die Betrachtung von urheberrechtlichen Aspekten erfordert das Vorhandensein der sogenannten Schöpfungshöhe. Diese ist rechtlich derart definiert, dass triviale Werke keinen urheberrechtlichen Schutz erlangen können. Im konkreten Kontext bedeutet dies, dass Trivialsoftware, die keine schützenswerten Eigenschaften aufweist, im deutschen Rechtsraum keinen urheberrechtlichen Schutz erlangen kann. Grundsätzlich dient ein derartiger Schutz der Sicherstellung der Verwertungsrechte des Autors, gemäß §§ 15 ff. des Urheberrechtsgesetzes (UrhG). So entsteht dieser Schutz bereits während des Verfassens des Quelltextes. Ein derartiger Vorgang wird im rechtlichen Sinne als Realakt bezeichnet und zielt dabei auf die Schöpfungshandlung als solche ab. Entsteht der Urheberschutz jedoch ohne weiteres Zutun als dem Realakt, so bedeutet dies praktisch, dass es keinerlei expliziter Kennzeichnung der Werke, also der Quelltexte, bedarf [8].

Software wird in Anlehnung an den zugrunde liegenden Quelltext als Sprachwerk im Sinne von § 2 Abs. 1 Nr. 1 UrhG. gesehen.

> **§ 2**
> **Geschützte Werke**
>
> (1) Zu den geschützten Werken der Literatur, Wissenschaft und Kunst gehören insbesondere:
>
> 1. Sprachwerke, wie Schriftwerke, Reden und Computerprogramme;
>
> [...]

Diese Sichtweise bestand jedoch keineswegs von Anfang an. In der historischen Entwicklung ging die allgemeine Lehre vielfach zunächst von einer Darstellung technischer Art aus, da Software im Sinne von Steuerungsbefehlen als Teil einer technischen Konstruktion verstanden wurde. Erst seit dem Jahr 1993 existiert der Abschnitt 8 der EG-Richtlinie 91/250/EWG[4], der explizit eine Sonderbestimmung für Computerprogramme vorsieht.

Durch diese Herausstellung von Software im Vergleich zu anderen Werken wurde jedoch ebenso die oben erwähnte Schöpfungshöhe als Voraussetzung zur Anerkennung als Werk erheblich herabgesetzt. Dennoch zeigt sich diese Neuregelung für den Autor einer Software durchaus als sehr positiv, da dieser im Rahmen der Ratifizierung der EG-Richtlinie im deutschen Rechtsraum nunmehr einen nicht ausschließbaren Vergütungsanspruch gemäß § 32 UrhG besitzt, sofern er die Urheberrechte innehat. Dies ist immer dann anzunehmen, wenn er die Schöpfungshandlung im eigenen Namen begangen hat.

Dennoch gab es auch aus dem OSS-Lager kritische Einwände. So erkannte das Institut für Rechtsfragen der Freien und Open Source Software (ifrOSS) die Möglichkeit von unkalkulierbaren Nachforderungen durch Softwareautoren [9]. Die sich anschließende Diskussion über mögliche wirtschaftliche Konsequenzen führte schließlich zu einer Abmilderung durch § 32 Abs. 3 Satz 3 UrhG.

[4] Richtlinie 91/250/EWG des Rates vom 14. Mai 1991 über den Rechtsschutz von Computerprogrammen

> (3) [...] Der Urheber kann aber unentgeltlich ein einfaches Nutzungsrecht für jedermann einräumen.

Das erklärte Ziel war somit, dass der Anwender vom Risiko etwaiger Nachforderungen befreit wird. Dass dieses Problem keinesfalls nur theoretischer Natur ist, belegt eine Einschätzung durch das Bundesministerium des Inneren im Rahmen des dort veröffentlichten Migrationsleitfadens [10] auf Seite 48. Bereits bei mittleren Projekten entsteht ein netzwerkartiges Geflecht von Autoren, welches eine Identifizierung und Zuordnung von Urheberrechten praktisch unmöglich macht. Es muss daher eine Zusammenfassung aller beteiligten Autoren erfolgen, die als Urhebergemeinschaft von Miturhebern verstanden wird. Dies stellt den einzig gangbaren Weg dar, sofern eine Partitionierbarkeit des Gesamtwerkes im Sinne einer wirtschaftlichen Verwertung nicht vorliegt. In praktischer Hinsicht bedeutet dies jedoch keinesfalls, dass dieser Kompromiss mit Nachteilen verbunden wäre. Vielmehr ist hier die OSD-Konformität der einzelnen Softwarelizenzverträge vorteilhaft, ist dadurch doch eine starke Ähnlichkeit der einzelnen Lizenzen im Sinne der Verwendung der Software festzustellen.

Im kommerziellen Sektor sind die Freiheiten bei der Verwendung von Software aus FOSS-Projekten sehr willkommen, die damit einhergehenden Pflichten jedoch eher unerwünscht. Eine Befreiung von den Pflichten kann zumindest insoweit nicht begründet werden, dass dem Verwender der Software die Lizenzbedingungen nicht bekannt waren. Die weitergehende Nutzung der Vervielfältigung, Verbreitung und Bearbeitung der Software erfordern die Zustimmung des Urhebers im Sinne einer Genehmigung der Ausübung dieser Handlungen oder alternativ eine Übertragung der Urheberrechte. Beide Varianten werden jeweils über Lizenzverträge zwischen den beiden Parteien vereinbart. Durch das Bestreiten der Kenntnis der Lizenz würde der Verwender der Software somit ebenso bestreiten, einen Lizenzvertrag mit dem Lizenzgeber geschlossen zu haben. Dies würde jedoch unmittelbar eine Urheberrechtsverletzung belegen [11].

Mittlerweile charakteristisch für die GPL ist deren stetige Entwicklung ähnlich eines Softwareproduktes und die Kennzeichnung des Entwicklungsstandes durch Versionsnummern. Der Urheber einer Software kann somit frei aus verschiedenen Versionen der GPL wählen oder sogar eine Lizenzierung anbieten, die sich über mehrere Versionen erstreckt. Fehlt eine explizite Festlegung der Lizenzversion durch den Lizenzgeber, darf der Lizenznehmer frei entscheiden, unter welcher Version er die Software nutzen möchte, gemäß Ziffer 9 Abs. 2 Satz 3 GPLv2 bzw. Ziffer 14 Abs. 2 Satz 3 GPLv3. Genau dies kann sich jedoch aus Sicht des lizenzgebenden Urhebers zu einem späteren Zeitpunkt als unerwünscht erweisen. Ein einmal gewährter Aufstiegspfad zu neueren Versionen oder die Wahlfreiheit des Lizenznehmers kann der Urheber dennoch nicht zurücknehmen, würde dies doch abstrakt einem Lizenzwiderruf entsprechen, der gemäß Ziffer 6 Satz 1 GPLv2 bzw. Ziffer 2 Abs. 1 Satz 1 GPLv3 verboten ist.

2.1.2 Patent- und Markenrechte

In der Vergangenheit vielfach diskutiert und bis heute in seiner Brisanz nicht minder relevant, ist das Problemfeld der Softwarepatente. Grundlage des Problems ist die regelmäßige Nichtpatentierbarkeit von abstrakt formulierten algorithmischen Verfahren. In diese Gruppe sind sowohl Algorithmen im informationstechnischen Sinne einzuordnen als auch mathematische Verfahren und Formeln. Da die Ersinnung derartiger Innovationen durchaus als geistiges Werk angesehen werden kann, kam es in der jüngeren Vergangenheit zu einer starken Forderung der Wirtschaft auf Neuregelung der Gesetzeslage und somit zur Einräumung einer unbeschränkten Patentierbarkeit. Die bisherige Situation erfordert die Vorstellung einer konkreten Implementierung der Innovation als Patentantrag, wodurch eine Abschwächung des

etwaig erteilten Patentes bezüglich der späteren wirtschaftlichen Verwertung vorgezeichnet ist. Wäre es möglich, die Innovation derart abstrakt patentieren zu lassen, dass ein darauf erteiltes Patent alle denkbaren konkreten Implementierungen einbeziehen würde, hätte dies existenzbedrohende Auswirkungen auf FOSS-Projekte. Die dort eingeräumte Einsehbarkeit des Quelltextes könnte von Patentinhabern problemlos als gerichtsfester Beleg dahingehend zweckentfremdet verwendet werden, dass gegen ein konkretes Patent verstoßen wurde [11].

Im unternehmerischen Umfeld kann den dort agierenden Parteien stets unterstellt werden, von einem Patent Kenntnis zu haben. Eine Berufung auf Unkenntnis käme dem Eingeständnis der Verletzung einer Sorgfaltspflicht gleich. Eben aus diesem Grund sehen besonders kleine und mittelgroße Unternehmen aus dem FOSS-Bereich in Softwarepatenten ein enormes Schadenspotential. Unter patentrechtlichen Aspekten könnten alle Parteien in Anspruch genommen werden, die an der Patentverletzung beteiligt sind. Konkret müsste also beispielsweise sowohl der beauftragte Lieferant oder Dienstleister haften als auch der Anwender jener Software, die patentierte Teile enthält. Besonders im privaten Umfeld ein unzumutbarer Zustand. Dem Patentinhaber steht es frei, gegen wen er seine Ansprüche geltend macht. So können die Distributoren betroffen sein, aber auch die Entwickler eines FOSS-Projektes, so dass die Existenz des Projektes unmittelbar auf dem Spiel stehen würde.

Das Patentrecht kann unter diesem Gesichtspunkt allerdings auch mit den Schutzzwecken von GPL-konformen Lizenzen in Konflikt geraten. So ist stets der gute Wille des Patentinhabers erforderlich. Um aus den Nachteilen eine geeignete Gegenstrategie abzuleiten, hilft die Zielsetzung des Patentrechts weiter. Geht es darum, echte Innovationen zu schützen und ihre wirtschaftliche Verwertbarkeit sicherzustellen, kann eine Lösung darin bestehen, den Innovationscharakter zu widerlegen. Patente werden im Sinne des deutschen Patentrechts nur dann erteilt, wenn es sich bei dem beantragten Verfahren um eine erstmalige Beschreibung handelt, es also von niemand anderem bereits ersonnen wurde. Hat ein FOSS-Projekt eine Innovation entwickelt, so gilt es, diese schnellstmöglich zu veröffentlichen und somit die Patentierbarkeit durch Dritte unmöglich zu machen. Dabei gilt es, den Erfordernissen bei der Beschreibung des Verfahrens im hinreichenden Maße nachzukommen. So ist eine wohldokumentierte Beschreibung vonnöten, aber auch die Verifizierbarkeit des Zeitstempels, um daraus den sogenannten Zeitrang eindeutig bestimmen zu können. Als wohldokumentiert wird nach herrschender Meinung der kommentierte und ggf. auch annotierte Quelltext des Verfahrens erachtet. Durch den Zeitstempel ist eine zeitliche Vergleichbarkeit mit späteren Dokumentierungen möglich, die entsprechend der Terminologie dann einen geringeren Zeitrang aufweisen und somit als nicht patentierbar zurückgewiesen werden.

Als organisatorisches Problem bleibt weiterhin die Verifizierbarkeit des Zeitstempels bestehen. Dieser muss von unabhängigen Institutionen nachprüfbar, manipulationssicher hinterlegt und mit dem Dokumentationsmaterial untrennbar verbunden sein. Entsprechend gibt es Initiativen, die sich mit der Frage der Einrichtung von Zentralstellen für FOSSP-Innovationen als Veröffentlichungsplattform beschäftigen. Die Vorteile derartiger Zentralstellen sind unmittelbar einsichtig. So kann eine hinreichende Frequentierung durch die Öffentlichkeit erreicht werden und die Institution als anerkannter Registrateur neuer Innovationen auch als Anlaufstelle für die nationalen Patentämter dienen [12]. Andererseits zwingt es die dezentral organisierten FOSS-Projekte in zentralisierte Strukturen hinein, die mit dem Gemeinschaftsgedanken nicht vereinbar sind. Dass es zu einer zeitnahen Lösung kommen muss, zeigt die inflationäre Genehmigungspraxis im Bereich Softwarepatente. Vielfach wird mit äußerst fragwürdigen Formulierungen und Ausnahmeregelungen operiert, die bei genauerer Betrachtung keinen Bestand hätten. Momentan ist die Rechtslage zumindest auf dem Papier eindeutig. Software wird im

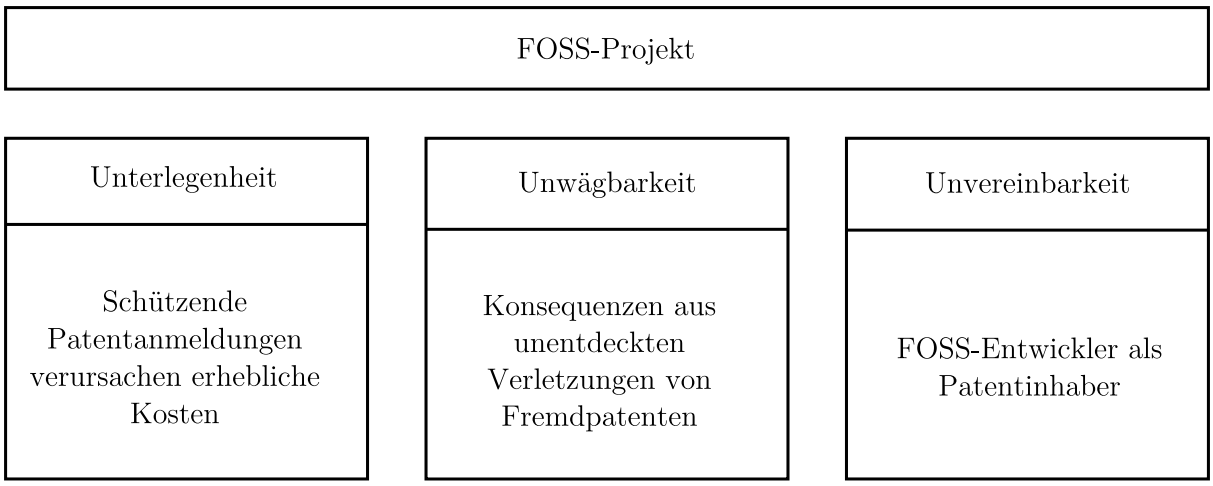


Abbildung 1: Die Problematik von Softwarepatenten.

Sinne von Programmen für Datenverarbeitungsanlagen gemäß Artikel 52 Abs. 2 Buchstabe c) Europäisches Patentübereinkommen[5] (EPÜ) als keine patentierbare Erfindung ausgewiesen.

Artikel 52
Patentierbare Erfindungen
[...]

(2) Als Erfindungen im Sinne des Absatzes 1 werden insbesondere nicht angesehen:

a) Entdeckungen, wissenschaftliche Theorien und mathematische Methoden;

b) ästhetische Formschöpfungen;

c) Pläne, Regeln und Verfahren für gedankliche Tätigkeiten, für Spiele oder für geschäftliche Tätigkeiten sowie Programme für Datenverarbeitungsanlagen;

[...]

Der vorläufige Stand der Entwicklung im Bereich Softwarepatente wird in [13] beschrieben. Problematisch bei der deutschen und europäischen Rechtslage ist insbesondere die unterschiedliche begriffliche Definition. Im deutschen Patentrecht wird das Verständnis bemüht, der technische Charakter sei maßgeblich, wobei Technik allgemein als die Beherrschung der Natur, ihrer Gesetzmäßigkeiten und Ressourcen verstanden wird. So ist die Steuerungssoftware beim Antiblockiersystem eines Kraftfahrzeuges oder jene bei Montagerobotern durchaus als wirkungswesentliche Komponente anzusehen, wodurch sie wiederum Patentschutz erlangt.

In der Praxis von FOSS-Projekten können sich Softwarepatente neben ihrer unmittelbaren bedrohlichen Eigenschaft auch als konzeptionell problematisch erweisen. So wirkt die Abkehr von finanziellen Zielstellungen unmittelbar als Nachteil, da die oben erwähnte proaktive Strategie und deren Umsetzung durch Beantragung von Sicherungspatenten mit erheblichen

[5] siehe auch unter `http://www.epo.org/patents/law/legal-texts/html/epc/2000/d/ma1.html`

Kosten verbunden ist. Des weiteren kann den meisten Projekten keine hinreichende Kompetenz im Umgang mit Fremdpatenten unterstellt werden, wodurch diese dem ständigen Risiko von unentdeckten Patentverletzungen ausgesetzt sind. Schließlich besteht für einen etwaigen Fremdpatentinhaber eben keine Verpflichtung zur Offenlegung, sofern die Ansprüche nicht in einem Gerichtsverfahren durchgesetzt werden. Den Projektentwicklern ist somit jede Möglichkeit genommen, Fremdpatente während der Entwicklung zu berücksichtigen und wirksam zu umgehen. Letztlich muss auch eine Unvereinbarkeit mit den Zielsetzungen von OSD-konformen Lizenzen festgestellt werden. Ein bekanntes Beispiel, wie ein einzelner Entwickler sich von der Gemeinschaft abkehrt und in Eigeninitiative ein Patent beantragt, wird in [14] vorgestellt.

Sicherlich lässt sich durch die Anpassung von OSD-konformen Lizenzen derartigen Problemen begegnen – nicht ohne Grund empfiehlt die FSF die Lizenzierung mit Aufstiegspfad auf zukünftige Lizenzversionen der GPL – allerdings konnte bisher nur beim Problemfeld der Unvereinbarkeit eine Verbesserung vorgenommen werden. So ist gemäß Ziffer 11 GPLv3 das Copyleft auch auf Patente anwendbar, allerdings bleibt das Problem der Sperrpatente weiter bestehen. In der Vergangenheit kam es mehrfach zu Situationen, in denen GPL-lizenzierte Software verwendet wurde, um die Entwicklungskosten konkreter proprietär vertriebener Software zu senken. Entgegen den auferlegten Pflichten, die sich aus der Lizenzannahme durch die weitergehende Nutzung ergeben, wurden die Änderungen und Erweiterungen an den Quellen zunächst nicht veröffentlicht, sondern stattdessen ein Patent auf diese Implementierung beantragt. Mit diesem Patent ging der Inhaber dann gegen Konkurrenten vor, die Software auf Basis der ursprünglichen GPL-lizenzierten Software verwandten. An diesem Beispiel wird die notwendige Unterscheidung zwischen Patent- und Urheberrechten besonders deutlich.

Als zweiter Komplex müssen auch die funktional geprägten markenrechtlichen Aspekte beachtet werden. Grundsätzlich spielt hierbei zunächst die Abgrenzung und Prominenz bei Marken eine große Rolle. Sie werden analog zu proprietären Herstellern dazu verwendet, sich von der Konkurrenz abzugrenzen oder die entgeltlichen nachgelagerten Dienstleistungen für potentielle Kunden attraktiver erscheinen zu lassen. Entsprechend stellt sich im Zusammenhang mit FOSS-Projekten die Verträglichkeit von Markenrechten mit den OSD-konformen Lizenzen. Wichtig hierbei ist die Unterscheidung zwischen Marken im eigentlichen Sinne, z. B. „Linux“, „Tux“ oder „Mandriva“ und den sogenannten Werktiteln, also Softwaretitel wie z. B. „Kaffeine“, „K3B“ oder „GIMP“. Deren Benutzung als Beschreibung der Beschaffenheit eigener Software ist erlaubt, z. B. „basiert auf ...“ oder „powered by ...“. Im kommerziellen Bereich spielt hierbei die Erkennbarkeit des Leistungserbringers, also jenem, der die entgeltlichen Dienstleistungen erbringt, eine wesentliche Rolle. Ferner muss sichergestellt sein, dass in der allgemeinen Wahrnehmung keine Verwechslung mit dem eigentlichen Markeninhaber zu befürchten steht. Ein Beispiel für den Namen einer Dienstleistung, die den genannten Voraussetzungen entspricht, wäre „KursLotse Linux-Neuling“. Die entgeltliche Dienstleistung besteht in diesem Fall in dem Anbieten von Einsteigerkursen für unerfahrene Anwender beim Umgang mit Linux-Betriebssystemen.

OSD-konforme Lizenzen sind in der Regel als markenneutral anzusehen, d. h. die einfache Nutzung der im Rahmen der Lizenznahme erworbenen Software erstreckt sich auch auf die darin zur Anwendung kommenden Titel und Logos. Die weitergehende Nutzung ist dagegen unzulässig. Der Titel der Software hat beim Vergleich mit anderen Projekten und ähnlicher proprietärer Software einen differenzierenden Charakter, eine Weiterverwendung durch Dritte würde diesem Zweck entgegenwirken. Die Verwendung von Marken und Werktiteln kommerziell agierender Unternehmen erfordert zumeist eine explizite Genehmigung, auch wenn man durch eine Lizenzierung die Software für die weitergehende Nutzung verwenden darf. Als Bei-

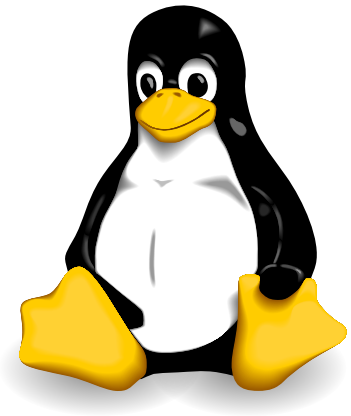


Abbildung 2: „Tux“ als Beispiel für eine in Deutschland nicht mehr eintragbare Marke. Der Begriff ist im allgemeinen Verkehr etabliert und hat somit einen gemeinverständlichen Bezug zum Betriebssystem Linux. Diese Eigenschaft gilt sowohl für die reine Wortmarke als auch für die obige Bildmarke. © Larry Ewing, Simon Budig and Anja Gerwinski.

spiele seien hier die Marken „MySQL“ und „Mozilla“ genannt, deren Verwendung im Rahmen von Mehrfachlizenzierungen berücksichtigt werden.

Analog zum Zeitrang bei Patenten gilt dieses Prinzip auch bei Marken. So ist es ausgeschlossen, dass ein Markenschutz für einen Namen gewährt wird, der als Softwaretitel vormals bereits Verwendung fand. Bei FOSS-Projekten stellt sich entsprechend das Problem, wie mit Dateien zu verfahren ist, die Marken enthalten, also z. B. die Grafik eines fremden Projektlogos. Sofern eine klare Trennung zwischen Quelltexten und markentragenden Dateien möglich ist, kann die Software ohne Einschränkung unter den Bedingungen der Nutzungslizenz verwendet werden. Für die markentragenden Dateien gilt dies dagegen nicht. Der Markeninhaber kann die Vervielfältigung, Verbreitung und Bearbeitung verbieten. Ist eine klare Trennung nicht möglich, so kann der Markeninhaber zur Genehmigung gezwungen werden, wenn er die Software Copyleft-basiert lizenziert hat. Insbesondere bei neuen Projekten findet in der Regel keine sorgfältige Existenzprüfung über bestehende Marken statt. So kann sich die Namenswahl als reales Risiko erweisen. Der verantwortliche Projektbetreuer des Vektorgrafikprogramms „Karbon14“ wählte ursprünglich den Namen „KIllustrator“ und zog sich sogleich den Unmut des proprietären Softwareherstellers Adobe Systems zu, der Inhaber der Marke „Adobe Illustrator“ war. Zwar ist die Abmahnung des Projektgründers unberechtigt gewesen, hätte jedoch die Hinzuziehung eines Rechtsbeistandes seitens des Projektbetreuers bedurft. Aufgrund fehlender finanzieller Mittel kam es zu keiner Gegenwehr, so dass schließlich das Projekt und die in dessen Rahmen entwickelte Software umbenannt wurden.

Im behördlichen Umfeld ist eine Risikoabwägung zwingend erforderlich. Insbesondere die Sicherstellung der Planungssicherheit bezogen auf die Wahrnehmbarkeit der erworbenen Nutzungsrechte ist von zentraler Bedeutung. Die von vielen proprietären Anbietern genannte Unvereinbarkeit OSD-konformer Lizenzen mit dem deutschen Recht wurde mittlerweile allerdings als obsolet und unzutreffend erkannt [10].

2.2 Vertragsrechtliche Aspekte

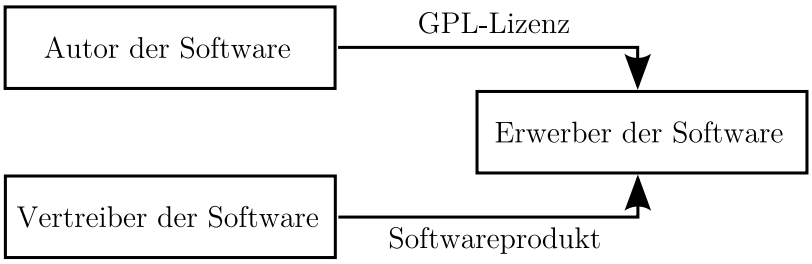
Stellvertretend für die Vielzahl OSD-konformer Lizenzen sei hier die oft zur Anwendung kommende GPL verwandt. Bei genauer Betrachtung handelt es sich bei konkreten Lizenzierungen stets um ein Dreiecksverhältnis zwischen den Parteien Urheber (Lizenzgeber), Anbieter (Distributor) und Benutzer (Empfänger), siehe auch Abbildung 3.

Das Softwareprodukt kann dabei durch entgeltliche Zusatzleistungen wirtschaftlich aufgewertet werden. So werden einzelne Linux-Distributionen durch hochwertiges Dokumentationsmaterial in Form von Handbüchern, physischen Datenträgern und Garantie- und Serviceleistungen aufgewertete und kommerziell vermarktet. Darüber hinaus generieren die Vertreiber derartiger Software ihre Gewinne durch Schulungen, kundenspezifische Anpassungen und Wartungsverträge. Die Erhebung von Entgelten für derartige Dienstleistungen, die auf OSD-konformer Software gründen, ist uneingeschränkt zulässig und wohl auch mitverantwortlich für den Erfolg von OSD-konformer Software im kommerziellen Bereich. Wesentlich hierbei sind die Vertragsverhältnisse, die zwischen den Leistungsempfängern und -erbringern geschlossen werden und neben der reinen Lizenznahme vom Urheber stehen. Dabei sind die lizenzrechtlichen Verhältnisse klar von den vertragsrechtlichen zu trennen, da der Urheber die Nutzungs- und Verwendungsrechte an der Software erteilt und der Anbieter wiederum Dienstleistungen erbringt, für die er haftet. Ebenso ist ein Zusammenfallen von Urheber und Anbieter denkbar, insbesondere dann, wenn es sich um Individualsoftware oder stark angepasste Standardsoftware handelt. In einem solchen Fall liegt in der Regel ein Werkvertrag zwischen Urheber und Benutzer vor, auf dessen Grundlage etwaige Haftungs- und Gewährleistungsansprüche des Benutzers begründet werden [8]. Für die rechtliche Bewertung der GPL stellt sich zunächst die Frage, wie diese in Bezug auf das Rechtsverhältnis zwischen dem Autor und dem Benutzer der Software zu bewerten und einzuordnen ist. In Betracht käme eine Einordnung als Softwarelizenzvertrag, als Schenkung oder als Gesellschaftsvertrag.

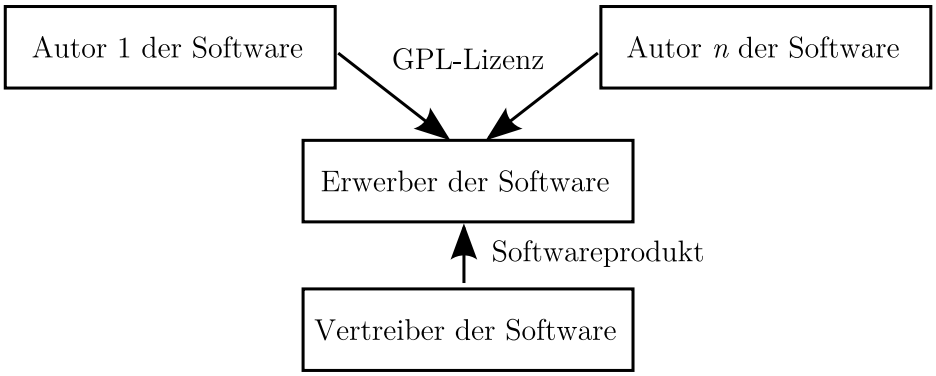
2.2.1 Vertragsverhältnisse

Bei Werk- und Dienstverträgen sind Haftungs- und Gewährleistungsfragen zumeist wohldefiniert und im Einvernehmen aller Vertragsparteien geregelt. So wird beispielsweise in einem Werkvertrag die Erstellung einer Software nach Vorgabe des Auftraggebers vereinbart. Dies geschieht zumeist auf Grundlage eines Pflichtenheftes und unter Einbeziehung einschlägiger Normen, wie der DIN 69905, die die Projektabwicklung im Detail regelt. Gegenstand von Dienstverträgen ist in der Regel die Erbringung von weitergehenden Dienstleistungen, wie der Konfiguration und Integration neuer Systeme in die bestehenden Betriebsabläufe, die Schulung des Personals oder die Erstellung von Dokumentationsmaterial. Ebenso können sicherheitsrelevante Dienste geregelt sein wie die Verifikation einer im Vorfeld definierten Beschaffenheit oder der Nachweis der Unbedenklichkeit und Kompatibilität gegenüber bereits bestehender Sicherheitsrichtlinien.

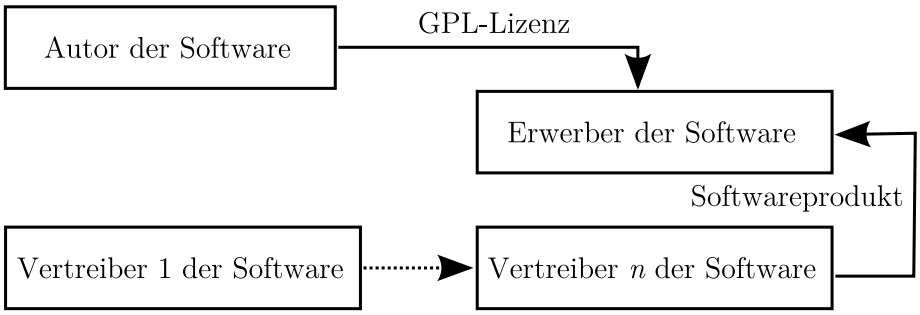
Der Versuch Standard- und Individualsoftware bezüglich ihrer Aufgabenorientierung zu unterscheiden, gestaltet sich häufig schwierig, da die zunehmende Anpassbarkeit von Standardsoftware an spezifische Problemstellungen einerseits als auch die funktional aufweitende Ergänzung von Spezialsoftware andererseits zur Lösung von Problemkomplexen oder bei der Konsolidierung als Lösungsansatz infrage kommen. Im Vergleich wird bei proprietären Softwarelizenzverträgen das Entgelt derart bemessen, dass Risiken beim Einsatz der Software als abgesichert gelten können. Eine derartige Absicherung ist bei OSD-konformen Lizenzen in



Lizenzerwerb von einem Autor



Lizenzerwerb von mehreren Autoren



Lizenzerwerb unabhängig vom Erwerb des Softwareproduktes

Abbildung 3: Typische Beziehungen bei der Lizenzierung durch die GPL. Ein Lizenzvertrag wird stets zwischen dem Autor der Software und dem Erwerber geschlossen. Von etwaig involvierten Anbietern erlangt der Erwerber lediglich das Softwareprodukt an sich.

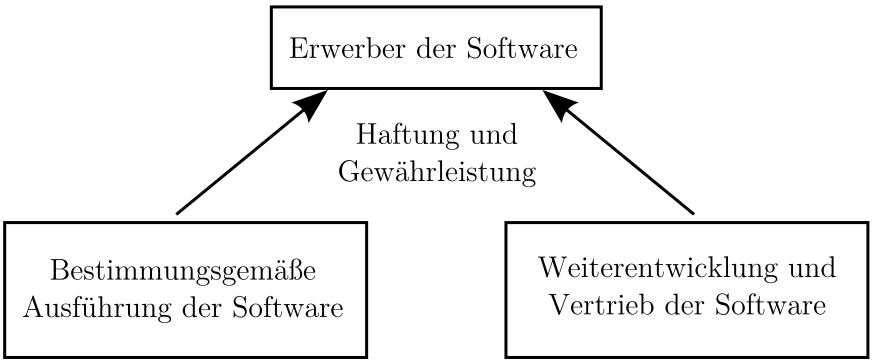


Abbildung 4: Haftungs- und Gewährleistungsfragen müssen sowohl unter dem Aspekt der bestimmungsgemäßen Ausführung als auch unter dem Aspekt der Weiterentwicklung und dem Vertrieb der Software betrachtet werden.

der Regel nicht gegeben. Das Risiko liegt in diesem Fall beim Lizenzgeber. Ein weiteres Risiko stellt die Wirksamkeit der erworbenen Nutzungsrechte dar, da bei einem Aufbauprojekt davon ausgegangen werden muss, dass die übernommenen Innovationen nicht patentrechtlich (siehe Kapitel 2.1.2 auf Seite 11) geschützt sind.

Eine weitere notwendige scharfe Abgrenzung muss zwischen der Gewährung der Verwendung der Software einerseits und der Gewährung von Entwicklungs- und Vertiebsrechten andererseits getroffen werden. Wie in Abbildung 4 dargestellt, müssen Haftungs- und Gewährleistungsfragen unter beiden Gesichtspunkten betrachtet werden [15].

Analog zu den vertragsrechtlichen Prinzipien im Zivilrecht gilt es zu untersuchen, ob es sich bei der GPL um einen Softwarelizenzvertrag handelt. Dazu ist es erforderlich, dass alle notwendigen Bestandteile vorhanden sind. Es bedarf zunächst eines Angebotes (Lizenzvergabe) und einer Annahme (weitergehende Verwendung der Software). Bereits an dieser Stelle tritt das Problem auf, dass das Angebot zur Lizenzierung an die Allgemeinheit gerichtet ist, da jedermann eine Lizenz vom Urheber der Software erhalten kann. Es fehlt ein konkreter Vertragspartner, der im klassischen Sinne unabdingbar ist. Dies dient dem Schutz des Anbieters, der gewöhnlich nicht beliebig oft leisten kann. Dem gegenüber steht, dass Software nahezu unbegrenzt vervielfältigt werden kann, so dass die Leistung des Lizenzgebers beliebig oft wiederholbar ist. Dass dieser jedermann tatsächlich eine Lizenz gewähren will, kann unterstellt werden, da die GPL genau dies vorsieht, gemäß Ziffer 6 Satz 1 GPLv2 bzw. Ziffer 5 Abs. 1 Buchstabe c) GPLv3. Allein die Verwendung der GPL als Lizenzierungsvariante kann insoweit als der Wille des Autors aufgefasst werden [8]. Die Lizenzvergabe gemäß der GPL ist weder befristet noch widerrufbar, eine Verweigerung zum Zeitpunkt der ersten Lizenzvergabe unmöglich geworden. Folglich gibt der Lizenzgeber eine Selbstverpflichtung ab, die Lizenzvergabe uneingeschränkt und unbefristet an jedermann zu ermöglichen. Aus der Sicht des Anwenders stellt die Ausführung GPLv2-lizenzierter Software hingegen noch keine Annahme der Lizenz dar. Nur die weitergehende Nutzung erfordert dies, gemäß Ziffer 0 Abs. 2 Satz 1 GPLv2. An dieser Stelle weist die Version 3 einen wesentlichen Unterschied auf, da gemäß Ziffer 2 Abs. 1 Satz 2 GPLv3 bereits die Handlung der Programmausführung nunmehr ebenso von der Lizenz geregelt wird.

2.2.2 Haftung und Gewährleistung

Grundsätzlich ist allen OSD-konformen Lizenzen gemein, dass die Frage, wer das Haftungsrisiko trägt, nur schwer zu beantworten ist. Ebenso ist offen, wer etwaig entdeckte Mängel in angemessener Frist behebt oder ob man die Nutzungsrechte an der Software durch Änderung der Lizenzierung wieder verlieren kann. Und schließlich ist es aus praktischen Überlegungen durchaus relevant, im Falle des Falles einen Ansprechpartner zu haben, der sich der Problemlösung annimmt. Wie im vorherigen Abschnitt bereits dargelegt, erfolgt der Einsatz von FOSS im behördlichen und unternehmerischen Bereich im Rahmen eines Dienstverhältnisses, so dass der Dienstleister die Risiken trägt und den Kunden betreut. Genau hierfür hat er mit dem Kunden ein Entgelt vereinbart, die Dienstleistungen somit als entgeltlich ausgewiesen.

Dem gegenüber steht die unentgeltlich eingeräumte Nutzung der Software durch den Urheber, wodurch sich auf vergleichbare Art keine Ansprüche des Anwenders der Software begründen lassen, der Extremfall der Arglist sei hierbei ausgeklammert. Dennoch kann der Anwender auf die Unterstützung einer weltweiten Nutzer- und Entwicklergemeinschaft zurückgreifen, sofern es sich um ein hinreichend großes Projekt handelt. Ein Blick in Richtung proprietärer Software würde den privaten Anwender ungleich schlechter stellen. So hat eine eventuell notwendige Mängelbehebung in Form einer Nachbesserung stets unentgeltlich zu erfolgen. Dennoch ist die Mängelbehebung durch den Hersteller der Software für den privaten Anwender zumeist kostenpflichtig. Mangels wettbewerblichem Druck oder durch Ausübung eines Quasimonopols besteht kaum die Bereitschaft zur Durchsetzung von Verbraucherinteressen. Besonders das im proprietären Bereich zu beobachtende Inverkehrbringen mängelbehafteter Software aufgrund von Termindruck oder voreiligen Zusagen hat den Begriff der Bananensoftware geprägt und beschreibt in bildlicher Weise das arglistige Schweigen einiger Softwarehersteller über die nicht gegebene Mängelfreiheit bei bestimmungsgemäßer Verwendung [11].

Der allgemeinen Rechtsprechung folgend, handelt es sich bei der GPL um einen sogenannten vorformulierten Softwarelizenzvertrag, der als allgemeine Geschäftsbedingung gemäß § 305 Abs. 1 BGB zu bewerten ist. Problematisch hierbei ist die Tatsache, dass es bis zum heutigen Tage keine offizielle Übersetzung der GPL ins Deutsche gibt, wodurch das Erfordernis der Verständlichkeit gemäß § 305 Abs. 2 Nr. 2 BGB fraglich ist.

§ 305
Einbeziehung Allgemeiner Geschäftsbedingungen in den Vertrag
[. . .]

(2) Allgemeine Geschäftsbedingungen werden nur dann Bestandteil eines Vertrags, wenn der Verwender bei Vertragsschluss

1. . . .
2. der anderen Vertragspartei die Möglichkeit verschafft, in zumutbarer Weise, die auch eine für den Verwender erkennbare körperliche Behinderung der anderen Vertragspartei angemessen berücksichtigt, von ihrem Inhalt Kenntnis zu nehmen,

[. . .]

Gerade im Falle der einfachen Nutzung im Sinne der ausschließlichen Programmausführung ist dies zu verneinen. Bei der weitergehenden Nutzung handelt es sich jedoch um eine

Handlung, die ein gewisses Maß an fachlicher Kompetenz erfordert, welche die Unterstellung der Verständlichkeit hinreichend begründet. Da gemäß GPL nur im zweiten Fall ein etwaiger Verstoß gegen die Lizenzpflichten gesehen und auch die nötige Fachkunde unterstellt werden kann, wird die GPL im Falle weitergehender Nutzung wirksamer Bestandteil des Softwarelizenzvertrages [8]. Ein weiteres Problem ergibt sich, wenn der Lizenzgeber nicht der Urheber ist. Einen gutgläubigen Erwerb von Nutzungsrechten sieht der Gesetzgeber nicht vor. Zwar ist es dem Urheber in einem solchen Fall nicht möglich, etwaige Ansprüche gegen den Lizenznehmer geltend zu machen. Allerdings steht es ihm frei, die Nutzung der Software im Sinne der vermeintlich erworbenen Lizenz zu untersagen. Diese Konstellation kann insbesondere für Aufbauprojekte existenzvernichtend sein.

Häufig verweist der Autor einer Software auf den allgemeinen Haftungsausschluss gemäß Ziffer 11 und 12 GPLv2 bzw. Ziffer 15 und 16 GPLv3. Als Bestandteile der allgemeinen Geschäftsbedingungen müssen derartige Regelungen den gesetzlichen Anforderungen von § 307 ff. BGB genügen. Im Detail ist den Regelungen zu entnehmen, dass gemäß Ziffer 11 Satz 2 GPLv2 bzw. Ziffer 15 Satz 2 GPLv3 jede Art von Haftung ausgeschlossen sein soll, also insbesondere auch bei sogenannten Rechtsmängeln. Grundsätzlich werden im Rahmen der Gewährleistung Ansprüche auf Mängelbehebung begründet, wogegen die Haftung Ansprüche auf Schadensersatz begründet. Da es an einer klaren gesetzlichen Regelung für OSD-konforme Lizenzen mangelt, können diese nur unter den allgemeinen Bestimmungen bewertet werden. Entsprechend ist ein genereller Haftungsausschluss, also insbesondere für vorsätzliches Handeln, im deutschen Rechtsraum unzulässig, gemäß § 307 Abs. 2 Nr. 1 BGB.

§ 307
Inhaltskontrolle
[. . .]

(2) Eine unangemessene Benachteiligung ist im Zweifel anzunehmen, wenn eine Bestimmung

1. mit wesentlichen Grundgedanken der gesetzlichen Regelung, von der abgewichen wird, nicht zu vereinbaren ist oder

[. . .]

Auch der salvatorische Charakter der Ziffern 11 Satz 1 und 12 Satz 1 GPLv2 bzw. Ziffern 15 Satz 1 und 16 Satz 1 GPLv3 kann in diesem Fall nicht weiterhelfen, da auch im internationalen Kontext ein derartiger Ausschluss unzulässig ist. Somit sind die genannten haftungsauschließenden Regelungen insgesamt unzulässig und auch nicht Bestandteil geschlossener Softwarelizenzverträge [8].

Sachmängel beschreiben alle Mängel einer Software, die ihr im Sinne eines Produktes zugeordnet werden können. Relevanz erlangt dies, wenn Software in kritischen Bereichen, wie zum Beispiel im medizinischen, behördlichen, industriellen oder volkswirtschaftlichen Sektor etc., verwendet wird. Ein Ausfall in vitalen Bereichen kann enorme Schäden jeglicher Art verursachen. Als begriffliches Gegenstück beschreiben Rechtsmängel zum Beispiel eine fehlende Urheberschaft des Lizenzgebers oder die unwirksame Übertragung der Nutzungsrechte an den Lizenznehmer. Konkret zu sehen in Kapitel 1.1 auf Seite 4 als Kontamination von FOSS-Projekten durch proprietären Quelltext. Ebenso wirkt auch das in Kapitel 2.1.1 auf Seite 10 erwähnte netzwerkartige Geflecht von Autoren, so dass die tatsächliche Urheberschaft nur

noch schwer rekonstruierbar oder gar nicht mehr feststellbar ist. Folgerichtig ist eine praktische Durchsetzung von Ansprüchen aus Rechtsmängeln in der Regel unmöglich, im speziellen Fall gemäß § 523 Abs. 1 BGB wohl nur bei arglistigem Schweigen möglich. Da es bei OSD-konformen Lizenzen allgemein meist keine Vereinbarung über konkrete Produkteigenschaften gibt, existiert auch keine Zusage der Verwendung der Software für bestimmte Zwecke durch den Autor, im Speziellen gemäß Ziffer 11 Satz 2 GPLv2 bzw. Ziffer 15 Satz 2 GPLv3. Somit ist die Sollbeschaffenheit unbestimmt und lediglich Schäden aus objektiven Mängeln, wie z. B. die Unbenutzbarkeit oder funktionale Mängel, relevant. Eine Entstehung von ersatzfähigen Mängeln kann nur durch die Ausführung der Software entstehen. Die Vervielfältigung, Verbreitung und Bearbeitung erfolgt stets auf Risiko des Lizenznehmers. Obwohl es durch die alleinige Programmausführung im Sinne der GPLv2 zu keiner Lizenzannahme kommt, muss eine Sachmängelhaftung in Betracht gezogen werden. Auch bei einer weitergehenden Nutzung der Software durch den versierten Anwender, ist von einer – wenn auch erheblichen selteneren – Ausführung der unmodifizierten Software auszugehen, bei der Schäden auftreten können. Da der Urheber bei einer GPL-basierten Lizenzierung lediglich die Nutzungsrechte an der Software durch Lizenzvergabe erteilt, schützt das UrhG nur den Urheber. Folglich kann der Lizenznehmer keinerlei Ansprüche aus der Lizenznahme ableiten. Im allgemeinen existiert nur ein OSD-basierter Softwarelizenzvertrag, wodurch daraus generell keine Schadensersatzansprüche begründbar sind. Somit kann festgestellt werden, dass trotz der Unwirksamkeit des generellen Haftungsausschlusses der GPL keine Haftung des Urhebers besteht. Diese Meinung vertritt auch [11].

2.3 Sonstige Aspekte

Insbesondere bei der weitergehenden Nutzung im kommerziellen Bereich durch Distributoren und Dienstleister müssen weitere Aspekte betrachtet werden, da deren Marktverhalten kartellrechtlichen Regeln unterliegt. So sind Preisabsprachen, wie dies im proprietären Bereich bereits mehrfach beobachtet werden konnte, durch das Copyleft-Prinzip unmöglich. Dies wiederum fördert im besonderen Maße den Wettbewerb, da alle auf den gleichen Voraussetzungen aufbauen. Erklärtes Ziel der OSD-konformen Lizenzen ist die Förderung einer kollektiv arbeitenden Entwicklergemeinde, bei der die Gleichstellung aller Beteiligten ein grundlegendes Prinzip ist. Dennoch kam es in der Vergangenheit zu Verstößen gegen die GPL, bei denen proprietäre Hersteller aus den sich daraus ergebenden Vorteilen ihren Nutzen ziehen wollten. Als Beispiel sei hier ein Fall aus dem Embedded-Bereich genannt, bei dem ein Gerätehersteller unter Missachtung der GPL Linux-basierte Router in Verkehr brachte. Die Käufer der Geräte erwarben ein mit Rechtsmängeln behaftetes Produkt, welches sie zwar problemlos bestimmungsgemäß verwenden konnten, der Weiterverkauf der gebrauchten Geräte jedoch zu einem Urheberrechtsverstoß geführt hatte, da kein wirksamer Softwarelizenzvertrag für das zur Anwendung kommende Linux-Betriebssystem vorlag. Der Hersteller informierte seine Kunden über diesen wesentlichen Produktmangel nicht, sondern täuschte diese durch arglistiges Schweigen. Eben dieses Schweigen verschaffte dem Hersteller einen deutlichen Wettbewerbsvorteil gegenüber der Konkurrenz, da er erhebliche Kosten durch die Verwendung von Linux einsparte.

Neben dem im privatwirtschaftlichen Bereich tätigen Unternehmen kommt es zuweilen vor, dass auch Behörden in unzulässiger Weise den Wettbewerb verzerren. Durch ihre Sonderstellung gegenüber reinen Privatgesellschaften genießen sie beim Verbraucher einen nicht unerheblichen – in letzter Zeit jedoch stark schwindenden – Vertrauensbonus. Empfehlungen

von Behörden oder gar behördliche Projekte können einen Verdrängungseffekt im freien Markt entfalten wie das Beispiel „Abrechnungs-Software für Zahnärzte“[6] eindrucksvoll belegt [12].

3 Beispiele aus der Praxis

In [16] wird auf nachvollziehbare Weise die Ansicht vertreten, dass die Übertragung bzw. Beibehaltung von kommerziellen Entwicklungsmethoden FOSS-Projekten zugute kommt, z. B. beim Mozilla-Projekt. Die Qualität und Produktivität hängen unmittelbar von der Größe der Projektgemeinde ab, die Agilität wiederum von der Stärke der Modularität eines Projektes und der Kompetenz des Entwicklerkerns.

3.1 Apache HTTP Server

Die Entwicklung am Apache HTTP Server begann im Februar des Jahres 1995. mit Erweiterungen für Webserver des National Center for Supercomputing Applications (NCSA) mit dem Namen *httpd*. Im Juli des selben Jahres begann man mit der Ersetzung der Codebasis durch eine Neuentwicklung, die im Januar 1996 als Apache *httpd 1.0* freigegeben wurde. Von Anfang an hatte das FOSS-Projekt mit dem Problem zu kämpfen, dass aufgrund der örtlich weit verstreuten Entwicklergemeinde eine Anpassung des Entwicklungsprozesses notwendig war. Die Arbeiten mussten koordiniert und die Kommunikation kanalisiert werden. Dem Rechnung tragend kam es zur Gründung der Apache Group (AG), die die Entwicklung des Servers fortan unter dem Projektnamen „Apache HTTP Server Project“ führte. Zumeist arbeiteten die beteiligten freiwilligen Entwickler neben ihrer offiziellen Anstellung am Projekt, so dass eine straffe Projektplanung mit wohldefinierten Aufgabenzuweisungen unmöglich war. Der Entscheidungsprozess war dezentral, die Kommunikation durch die Verwendung von E-Mail-Listen asynchron und es existierte nur ein rudimentäres Wahlsystem um Konflikte zu bereinigen. Die Aufgabenverteilung erfolgt bis heute nach dem Rollensystem:

1. **AG-Mitglieder** — Um ein Mitglied der Apache Group zu werden, ist die Beteiligung am Projekt über eine Mindestdauer von sechs Monaten vonnöten. Zusätzlich kann nur Mitglied werden, wer auf Vorschlag von den anderen Mitgliedern gewählt wurde. Ein Mitglied bekommt Lese- und Schreibzugriffe auf das Versionsverwaltungssystem und darf an allen Stellen den Quelltext modifizieren, löschen oder neuen Quelltext hinzufügen. Eine freiwillige Selbstbeschränkung auf bestimmte Systemteile ist jedoch erwünscht. Abstimmungen finden nur bei umfassenden Änderungen statt, um den Kommunikationsaufwand gering zu halten.

2. **Kernentwickler** — Der Kreis von aktiven Entwicklern, aus denen neue Mitglieder nominiert werden. Die Kernentwickler arbeiten nach keinem definierten Entwicklungsprozess. Ihre Aufgabe besteht in der Identifizierung von Problemen, der Formulierung neuer Funktionen und der Auslagerung von anstehenden Arbeiten an freiwillige Entwickler. Alle eingehenden Lösungsvorschläge aus dem Kreis dieser freiwilligen Entwickler werden von den Kernentwicklern beurteilt und ggf. als geeignete Lösung bestimmt. Darüber hinaus obliegt ihnen die Aufgabe, neue Quelltexte zu verfassen oder extern zufließende zusammenzuführen. Neue Quelltexte werden grundsätzlich zuerst auf lokalen Kopien der aktuellen Entwicklerversion auf ihre Verträglichkeit und Stabilität geprüft, bevor sie

[6] BGH, I ZR 174/91

den AG-Mitgliedern vorgelegt werden. Nach entsprechender Freigabe spielen die Kernentwickler neuen Quellen und Dokumentationsmaterial in die Versionsverwaltung ein. Automatische E-Mail-Listen informieren alle beteiligten Entwickler über aktuelle Änderungen.

3. **Freiwillige Entwickler** — Jeder kann sich ins Apache HTTP Server Projekt einbringen. Einen Aufstieg im Rollensystem erlangt man durch Erwerb entsprechender Reputation.

Sowohl die Entwickler- als auch die Anwendergemeinschaft kommuniziert über E-Mail-Listen, BUGDB und Diskussionsforen wie dem USENET. Die beiden letztgenannten werden für Fehlerberichte verwendet, die E-Mail-Listen dagegen vorwiegend für Änderungsvorschläge. Um die Kommunikationskanäle effizient zu nutzen, werden diese zusätzlich priorisiert, wobei die E-Mail-Listen der Entwickler die höchste Priorität genießen. Kritische Probleme müssen als solche durch die Mehrheit der AG-Mitglieder bestimmt werden, da diese eine Veröffentlichung bis zu deren Behebung stoppen können. Die Entwicklergemeinde nutzt zur Verteilung der Verantwortung das Prinzip der impliziten Eigentumsrechte an Systemteilen. So fühlt sich jeder Kernentwickler für bestimmte Systemteile verantwortlich, neue Kernentwickler suchen sich selbstständig eigene Teile. Kommt es dennoch zu einem Streitfall sind diese Eigentumsrechte jedoch irrelevant und der Streit wird durch Mehrheitsentscheid der AG-Mitglieder entschieden.

Neben vorgelagerten Testläufen, die auf den bereits genannten lokalen Arbeitskopien der Entwickler auf ihre Verträglichkeit geprüft werden, kommen auch sogenannte Inspektionen zum Einsatz, die eine stichprobenartige ausführliche Sichtung der Quelltexte darstellen. Zusätzlich werden durch die Kernentwickler Bereichsüberprüfungen (unit tests) durchgeführt. Es besteht einerseits die Möglichkeit Änderungen direkt in die Versionsverwaltung einzuspielen oder andererseits Korrekturpakete zu erstellen, die nur die geänderten Quelltextbereiche enthalten. Jede Änderung an den stabilen Veröffentlichen unterlaufen eine zusätzlichen Gegenprüfung (review), wohingegen Änderungen an den Entwicklerversionen zuerst eingespielt und dann gegengeprüft werden. Jede einzelne Einspielung in die Versionsverwaltung wird allgemein über die CVS-E-Mail-Listen bekannt gegeben, die die Kernentwickler in der Regel überprüfen. Es ist auch registrierten Außenstehenden möglich und sogar erwünscht, Gegenprüfungen vorzunehmen. Auf diese Weise können Feldtests durchgeführt werden, bevor eine neue Version offiziell veröffentlicht wird. Da die Veröffentlichung selbst mehr ist als die bloße Bereitstellung über Bezugsquellen, wird jeder einzelnen Veröffentlichung ein freiwilliger Kernentwickler zugeordnet, der den Vorgang koordiniert, protokolliert und kontrolliert.

Von außen betrachtet lässt sich feststellen, dass der äußere Bereich der freiwilligen Entwickler vorwiegend im Bereich der Mängelbehebung tätig ist, der Innenbereich der Kernentwickler und AG-Mitglieder sich dagegen um die Weiterentwicklung des Projektes kümmert. Rückblickend hat sich die Arbeitsweise des Apache HTTP Server Projektes als optimal für kleine Projekte erwiesen. Der Server selbst ist sehr kompakt entworfen und überschaubar gestaltet. Seine Komplexität, die er häufig in der Praxis zeigt, erreicht er durch zahllose Erweiterungen, die in externen Projekten entwickelt werden.

3.2 Mozilla Foundation

Wie einige andere Projekte auch, hat Mozilla seine Wurzeln bei einem vormals kommerziellen Projekt. Die Entwicklung als FOSS-Projekt begann mit der Freigabe der Quellen durch Netscape im Januar des Jahres 1998. Das erklärte Ziel dabei war, einen ähnlichen Erfolg und

vergleichbare Popularität wie Linux zu erreichen. Um die Entwicklung besser koordinieren zu können, wurde am 23. Februar des selben Jahres die Mozilla Organization gegründet, die am 15. Juli 2003 nach dem Rückzug von AOL in der mit Spendenmitteln neugegründeten Mozilla Foundation aufging. Als kommerzieller Ableger wurde am 3. August 2005 schließlich die Tochtergesellschaft Mozilla Corporation gegründet.

Koordiniert und kontrolliert wird das FOSS-Projekt durch einen Stab von Mitarbeitern, dem *mozilla.org staff*. Auch bei diesem Projekt gibt es eine Einteilung nach Aufgaben. Eine Gruppe kümmert sich um die Betreuung der Anwender- und Entwicklergemeinde mit der Zielsetzung Qualitätssicherung, die zweite um die Koordination der Meilenstein-Veröffentlichungen. Die dritte Gruppe bearbeitet als Aufgabenschwerpunkt die Erstellung von Werkzeugen und die Instandhaltung der Projektinfrastruktur, die letzte Gruppe betreut das öffentliche Mängelverzeichnis Bugzilla. Das Projekt kann auf bezahlte Vollzeit-Entwickler zurückgreifen, wobei für die einzelnen Systemteile Entscheidungsträger (Eigner) aus der Entwicklergemeinde bestimmt werden. Zugang zur Versionsverwaltung erhalten einzelne Entwickler nur nach Erlangung entsprechender Reputation. Der Eigner eines Systemteils kann die Erlaubnis zum Hinzufügen oder Ändern von Dateien und Verzeichnissen erteilen, das Hinzufügen neuer Systemteile erfordert dagegen die Zustimmung des Stabes. Dort liegt auch die letzte Entscheidungsgewalt, wenn es zu Konfliktfällen gekommen ist. Dem Stab kommt ebenso Aufgabe zu, Eigner zu entlassen oder neue zu ernennen. Die Projektleitung veröffentlicht in regelmäßigen Abständen einen aktualisierten Projektplan (roadmap), auf dem die Meilensteine nebst inhaltlicher und zeitlicher Vorgaben aufgeführt sind. Einer derartigen Veröffentlichung geht im Vorfeld stets eine Einbeziehung und Konsultation der Entwicklergemeinde voraus. Das öffentliche Mängelverzeichnis Bugzilla steht jedem offen. So darf man ein eigenes Konto einrichten, über Fehler berichten und Verbesserungsvorschläge unterbreiten. Dabei sollte man im Interesse der Entwickler, die die Berichte begutachten, den eigenen Bericht so präzise und knapp wie möglich verfassen. Charakteristisch für das FOSSP ist die weitgehend selbstkoordinierende Arbeitsweise, bei der jeder Entwickler sich Probleme oder neu zu implementierende Funktionen aussuchen darf. Es gilt lediglich die Maxime, dass jeder das machen soll, worin er nach eigener Einschätzung am besten ist. Um eine Art Hilferuf arbeitender Entwickler an die Gemeinschaft einzurichten, gibt es das Schlüsselwort „help-wanted“. Andere Entwickler sind eingeladen, an derart markierten Einträgen ihr Hilfe anzubieten und Vorschläge zur Problemlösung vorzustellen. Für den flüssigen und regen Ideenaustausch werden vorwiegend Diskussionsforen verwendet.

Für vorgelagerte Testläufe werden täglich Entwicklerversionen (daily builds) zusammengestellt, die auf den gängigsten Plattformen getestet werden. Die dort entdeckten Mängel unterbrechen sogleich die Weiterentwicklung. Auf diese Weise kommt es zu einer sofortigen Behebung kritischer Defekte, da die Weiterentwicklung nicht möglich ist, solange der Fehler besteht. Aufgrund der Komplexität der Testszenarien findet zusätzlich eine funktionale Trennung der Überprüfungen statt, die wiederum durch eine Einteilung in Arbeitsgruppen (test teams) durchgeführt wird. Das domänenspezifische Wissen der Systemteileigner wird für die Inspektion der einzelnen Module verwandt. Integrationsbezogene Inspektionen, die sich mit dem Zusammenwirken der einzelnen Module zueinander auseinandersetzen, werden von übergeordneten Entwicklern durchgeführt. Somit ist die Verträglichkeit einzelner Korrekturen mit dem Gesamtsystem sichergestellt. Als stetiger Prozess ist die Versionserstellung (build process) konzipiert. Jedes Systemteil wird mit Freigabeattributen versehen, die sich auf konkrete Plattformen und konkrete Versionen beziehen. Neben den täglichen Schnappschussversionen (nightly builds) werden zusätzlich im Monatsrhythmus Meilensteinversionen (milestone builds)

veröffentlicht. Die Systemteileigner[7] werden ergänzend durch gleichgestellte Entwickler (peers) unterstützt.

Die Beherrschbarkeit großer Projekte hängt unmittelbar von einer guten Koordination ab, wobei die Systemstruktur als Grundlage dienen kann, die Aufgaben und Zuständigkeiten gleichmäßig und zielgerichtet zu verteilen. Das Projekt selber arbeitet derzeit an einer Sicherheitsmetrik anhand des Internetbrowsers Firefox. Die im Rahmen der Entwicklung von Firefox gewonnenen Erkenntnisse sollen auch bei anderen Projekten Anwendung finden. Ziel ist dabei eine Quantifizierung der Effektivität der Sicherheitsbemühungen und des Risikos für den Anwender. Das Verfahren ist als eine sogenannte Verlaufsmetrik konzipiert, die Betrachtungen und Beobachtungen entlang einer Zeitachse ermöglicht und Veränderung unmittelbar aufzeigt. So sollen sowohl Verbesserungen als auch Verschlechterungen aufgezeichnet und die Bestimmbarkeit von Reaktionszeiten ermöglicht werden [17].

4 Fazit

Die Entwicklung und der Vertrieb bei FOSS-Projekten erfolgt zumeist dezentral und selbstorganisierend. Es existiert in der Regel kein exklusiver Hersteller oder Anbieter, so dass sowohl die Bezugsquelle als auch der Urheber als Anlaufpunkt in Betracht kommen. Die OSD ist im Sinne einer Rahmenlizenz auch als Vorlage und somit als Rahmenwerk für die Formulierung unterschiedlichster Lizenzierungsformen verwendbar. Dennoch ist weiterhin eine Einteilung möglich, da OSD-konforme Lizenzen sowohl Copyleft-behaftet als auch Copyleft-unbehaftet sein können. Die Idee, die hinter der OSD steht, war vormals ideologisch und dogmatisch geprägt. Heutzutage stellt sie ein etabliertes und konkurrenzfähiges Vermarktungsmodell dar. Durch den enormen Innovationsdruck, fairen Wettbewerb im Sinne des Terminus „Krieg der Projekte“ und durch das Infragestellen proprietärer Vermarktungsmodelle ist sie als gangbare und vollwertige Alternative anzusehen.

Allgemeine Aussagen zu Haftungsfragen sind kaum möglich, dazu unterscheiden sich die nationalen Rechtsordnungen zu stark. Derartige Fragestellungen müssen stets im Kontext einer geltenden Rechtsordnung erörtert werden. Allgemeine Ausschlussklauseln wie sie die GPL verwendet sind dabei als kritisch einzustufen und stellen ein vermeidbares Risiko an Unvereinbarkeit dar. Der generelle Haftungsausschluss der GPL erweist sich sowohl in der deutschen Rechtsordnung als auch im Kontext des internationalen Privatrechts als unzulässig. Dennoch muss sich der redliche Urheber keinem Haftungsrisiko ausgesetzt sehen, da Art und Umfang derart unbestimmt sind, dass Zusicherungen über bestimmte Produkteigenschaften oder die Tauglichkeit der Software für bestimmte Verwendungen durch OSD-konforme Lizenzen in aller Regel nicht zugesichert werden.

Auf der anderen Seite bestehen jedoch sekundäre Risiken für FOSS-Projekte, aus denen sich durchaus Haftungsszenarien entwickeln können. So müssen markenrechtliche Aspekte unbedingt beachtet werden, aber auch patentrechtliche Fragen im Sinne von Softwarepatenten können keineswegs als abschließend beantwortet angesehen werden. Dafür ist der Druck aufgrund des starken Interesses der proprietären Hersteller an derartigen Patenten viel zu groß. Eine gangbare, vielleicht letztlich sogar zwingende Strategie stellt die Selbstorganisation und -koordination der globalen Gemeinschaft dar. Der Schutz des Kollektivs kann nur durch das Kollektiv selber erfolgen, indem es die Schaffung von freien Innovationsgremien forciert, die

[7] Liste der aktuellen Systemteileigner abrufbar unter `http://www.mozilla.org/owners.html`

das globale geistige Eigentum vor der Gefahr von Sperrpatenten schützen und im Gegenzug die Durchsetzung von Lizenzpflichten vorantreiben, z. B. bei Verstößen gegen die GPL.

Literatur

[1] Mark Tolliver. *Open Source Risks and Responsibilities*. OS news, September 2007. `http://www.osnews.com/story/18610/Open_Source_Risks_and_Responsibilities`.

[2] Robert Gobeille. *FOSSology and FOSS Governance*. The FOSSBazaar Project, Juli 2008. `https://fossbazaar.org/?q=filemanager/active&fid=133`.

[3] Institut für Rechtsfragen der freien und Open Source Software. *Open Content und Open Access*. September 2008. `http://www.ifross.de/ifross_html/opencontent.html`.

[4] Karen M. Sandler Bradley M. Kuhn, Aaron Williamson. *A Practical Guide to GPL Compliance*. Software Freedom Law Center, August 2008. `http://www.softwarefreedom.org/resources/2008/compliance-guide.pdf`.

[5] Jeffrey S. Hammond. *Enterprise Open Source Use In 2007*. Forrester Research, Inc., Juni 2008. `http://www.forrester.com/Role/Research/Workbook/0,9126,46115,00.html`.

[6] Gail C. Murphy John Anvik, Lyndon Hiew. *Who Should Fix This Bug?* Association for Computing Machinery, Mai 2006.

[7] John Anvik. *Automating Bug Report Assignment*. Association for Computing Machinery, Mai 2006.

[8] Olaf Koglin. *Opensourcerecht*. Peter Lang – Internationaler Verlag der Wissenschaften, 2007. ISBN 3-63156-308-6.

[9] Institut für Rechtsfragen der freien und Open Source Software. *Stellungnahme des ifrOSS zu den Vorschlägen für eine Regelung des Urhebervertragsrechts*. `http://www.ifross.de/ifross_html/urhebervertragsrecht.pdf`.

[10] Bundesministerium des Innern. *Migrationsleitfaden Version 3.0 – Leitfaden für die Migration von Software*. Bundesministerium des Innern, 2008. `http://www.kbst.bund.de/`.

[11] Institut für Rechtsfragen der Freien und Open Source Software. *Die GPL kommentiert und erklärt*. O'Reilly Verlag, 2005. ISBN 3-89721-389-3.

[12] Dr. Axel Metzger Dr. Till Jaeger. *Open Source Software – Rechtliche Rahmenbedingungen der Freien Software*. Verlag C. H. Beck, 2006. ISBN 3-406-53803-7.

[13] Stefan Krempl. *Die Auseinandersetzung um das Patentwesen und der Streit über Softwarepatente*. Heise Zeitschriften Verlag, Juni 2005. `http://www.heise.de/ct/hintergrund/meldung/61230`.

[14] Axel Metzger. *Kollision in Echtzeit – RTLinux und das U.S.-Patent 5,995,745*. Linux New Media AG, Februar 2003. `http://www.ifross.de/ifross_html/art16.html`.

[15] Carsten Schulz. *Dezentrale Softwareentwicklungs- und Softwarevermarktungskonzepte; Vertragsstrukturen in Open Source Modellen*. Carl Heymanns Verlag, 2005. ISBN 3-452-25965-X.

[16] James D. Herbsleb Audris Mockus, Roy T. Fielding. *Two Case Studies of Open Source Software Development: Apache and Mozilla*, volume 11. Association for Computing Machinery, Juli 2002.

[17] Mozilla Security Blog. *Mozilla Security Metrics Project.* Mozilla Project, August 2008. `http://blog.mozilla.com/security/2008/07/02/mozilla-security-metrics-project/`.

A Begriffsdefinitionen

Aufbauprojekt Projekt, welches auf den Quellen eines oder mehrerer FOSS-Projekte aufbaut und diese auf Grundlage OSD-konformer Lizenzen nutzt. Das Aufbauprojekt muss nicht notwendigerweise selbst ein FOSS-Projekt sein.

Aufstiegspfad Findet im Kontext der Versionierung der GPL Anwendung. Er beschreibt die Möglichkeit des Urhebers, seine Software unter einer bestimmten Version zu veröffentlichen jedoch gleichzeitig auch spätere Versionen für die Verwendung durch den Lizenznehmer zuzulassen. Ein einmal gewährter Aufstiegspfad kann nachträglich nicht widerrufen werden.

Binärfassung Eine maschinenlesbare ausführbare Übersetzung der Quelltexte eines Softwareproduktes.

Defekt Semantischer, selten auch syntaktischer Fehler im Quelltext, der durch Sichtung, Gegenprüfung oder Einsatz der Software entdeckt wird.

Entität Gesamtheit eines Werkes, die als Vielgestalt zu verstehen ist. So kann ein Werk Eigenschaften aufweisen, die es charakterisieren oder es in essentieller Weise beschreiben. Nicht jede Eigenschaft muss notwendigerweise offensichtlich oder bekannt sein. Vielmehr können sich einzelne Eigenschaften erst in einem konkreten Kontext offenbaren.

Mangel Allgemein jede Abweichung von den erwarteten Produkteigenschaften. Konkret auf Softwareprodukte bezogen, ist deren Gesamtheit gemeint. D.h. ein Mangel kann überall auftreten, z. B. im Quelltext, in der Dokumentation, der grafischen Benutzeroberfläche oder sich auf Ergonomie und Lokalisierungsaspekte beziehen.

Quelltext Menschenlesbare Abfolge von Befehlen in einer höheren Programmiersprache mit begleitenden Kommentaren und Annotationen. Insbesondere zur Hervorhebung der Lesbarkeit und Verständlichkeit dieser Repräsentation eines Softwareproduktes wird das Suffix „-text“ anstelle der weit verbreiteten Alternative „-code“ verwendet.

Rechtsinhaber Jene Person oder Gesellschaft, die die Verwertungsrechte an einem Werk (Softwareprodukt) inne hat. Ein Programmierer im Sinne eines Autors ist nur Rechtsinhaber, wenn er das Werk im eigenen Namen erschaffen hat, also z. B. nicht als Mitarbeiter eines Unternehmens.

Salvatorischer Charakter Im juristischen Sinne eine funktionale Eigenschaft einer Regelung. Eine derartige Regelung greift, wenn sich Teile eines Vertrages oder Geschäftsbedingungen nicht mit geltendem (nationalen) Recht vereinbaren lassen. Sie stellen im

Sinne einer Schadensbegrenzung sicher, dass die Wirksamkeit nichtbetroffener Teile erhalten bleibt.

Weitergehende Nutzung Im Rahmen von OSD-konformen, im speziellen GPL-basierten, Softwarelizenzverträgen wahrgenommene Nutzungsrechte der Vervielfältigung, Verbreitung und Bearbeitung der Quellen.

B Semantische Übersetzungstabelle

build process	Versionserstellung
code copyright	Urheberrechte am Quelltext
code freeze	Versiegelung des Quelltextes
code owner	Quelltexteigner
contributor	Unterstützer
commit	Einspielung, einspielen
content	geistiges Werk
copyright notice	Urheberschutzvermerk (Hinweis auf den Schutz durch den United States Copyright Act)
core developers	Kernentwickler
dead code	stillgelegte Systemteile
defect	semantischer Fehler im Quelltext
driver	Koordinator
dual licensing	Mehrfachlizenzierung
members	Mitglieder (z. B. Apache-Group-Mitglieder)
module owner	Systemteileigner (Verantwortlicher eines Systemteils)
news group	Diskussionsforum
nightly build	Schnappschussversion
open bug repository	öffentliches Mängelverzeichnis
patch	Nachbesserung, Korrektur
peer	gleichgestellter Entwickler
release	offizielle Veröffentlichung, Freigabe
reporter	Berichterstatter
resolver	Instandsetzer
roadmap	Projektplan
showstopper	kritisches Problem
source code	Quelltext (menschenlesbar)
submitter	Berichterstatter
test team	Arbeitsgruppe der Qualitätssicherung
triager	Sortierer
prerelease testing	vorgelagerte Testläufe
release manager	Verantwortlicher der Veröffentlichung
review	Gegenprüfung
volunteers	freiwillige Entwickler