Zertifizierung von Open Source Lieferketten

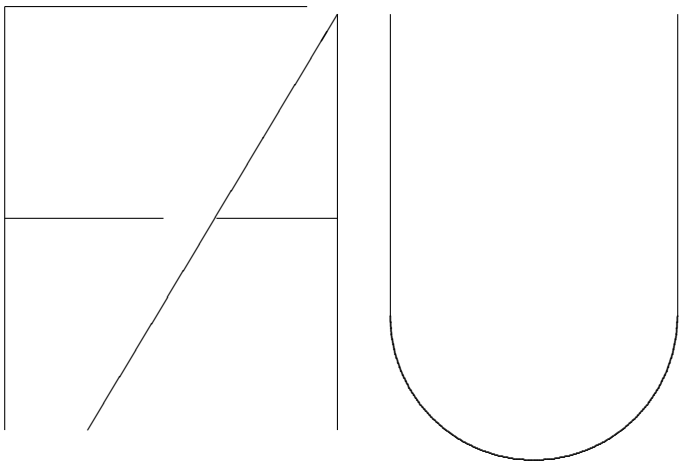
Masterarbeit

Benedikt Lempetzeder

Professur für
Open-Source-Software

Department Informatik
Technische Fakultät

Friedrich Alexander-
Universität
Erlangen-Nürnberg



Zertifizierung von Open Source Lieferketten

Masterarbeit im Fach Informatik

vorgelegt von

Benedikt Lempetzeder

geb. 28.04.1988 in Lichtenfels

angefertigt am

Department Informatik
Professur für Open-Source-Software
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Prof. Dr. Sc.-Techn. Dirk Riehle, M.B.A.

Beginn der Arbeit: 14.04.2013
Abgabe der Arbeit: 14.10.2013

Erklärung zur Selbständigkeit

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch die Professur für Open-Source-Software, wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Masterarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 14.10.2013

(Benedikt Lempetzeder)

Kurzfassung

Zertifizierung von Open Source Lieferketten

Open Source Software ist aus der heutigen Softwareentwicklung nicht mehr wegzudenken. Praktisch jedes Softwareprodukt enthält Teile von Open Source oder wird gänzlich als Open Source Software veröffentlicht. Wie bei anderen Softwareprodukten wird diese auch mit Hilfe einer komplexen Lieferkette gefertigt. Innerhalb dieser Lieferkette bestehen viele Risiken, die den Erfolg des Produktes auf juristische, ökonomische und gesellschaftliche Weise gefährden können. Um diese Risiken unter Kontrolle zu bringen und die Auswirkungen abzuschwächen, setzen Unternehmen verschiedene Maßnahmen um.

Durch das Einführen eines Zertifizierungsprozesses, der die Implementierung der Vorkehrungen gegen Open Source Risiken analysiert, wird die Auswahl der Zulieferer in der Lieferkette transparent gemacht. Als Ziel davon sollen die Risiken in der Lieferkette auf ein Mindestmaß reduziert werden können.

Die Zertifizierung basiert auf einem zweidimensionalen Modell, das die Art der Nutzung von Open Source Software und die Qualität der Reduzierung der Risiken betrachtet. Das Ziel ist die umfassende und ganzheitliche Zertifizierung der Lieferkette eines Softwareproduktes, um sowohl Transparenz zu fördern, Risikoverminderung voranzutreiben als auch Selbstreflexion zu erlauben.

Abstract

Certification of Open Source Supply Chains

Today no software development wants to go without Open Source. Practically every software product comes with parts of Open Source or is released completely as Open Source software. Like other software products Open Source is also built with a complex supply chain. Within the supply chain lots of risks are prevailing, which are influencing the success of the product in a judical, economical and social manner. Companies have to implement provisions to get these risks under control and to weaken their effects.
A certification process is introduced, which analyzes the implementation of these precautions to enable transparency in the selection process of suppliers. As a consequence of the implementation all risks can be reduced to a minimum.
The certification is based on a two dimensional model, which considers the type of use of Open Source and the quality of the reduction of the risks. The goal is the extensive and holistic certification of the whole supply chain to ensure transparency, risk reduction and self-reflexivity.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BSD Berkeley Software Distribution

CMM Capability Maturity Model

FAT File Allocation Table

GPL GNU General Public License

ISO International Organization for Standardization

IT Informationstechnologie

LGPL Lesser GNU General Public License

MIT Massachusetts Institute of Technology

MPL Mozilla Public License

NASA National Aeronautics and Space Administration

SPDX Software Package Data Exchange

SPICE Software Process Improvement and Capability Determination

USA United States of America

1 Einführung

Open Source Software verliert zusehends das Prädikat eines Randphänomens in der Softwareentwicklung. In der heutigen Softwareentwicklung ist der Einsatz von Open Source Software nicht mehr wegzudenken. Bereits rund 80% der Entwicklungsfirmen benutzen Open Source Software, um ihre Software zu erstellen [Rie11].
Der Einsatz von Open Source Software kann unterschiedlich geprägt sein. Zum Einen kann sich der Gebrauch auf den Einsatz von Open Source Software als infrastrukturelle Software beschränken. Unter dieses Gebiet fällt zum Beispiel das Nutzen von Open Source Entwicklungsumgebungen, Open Source Compilern oder das Binden von Open Source Programmbibliotheken. Zum Anderen kann Open Source Software in ein komplexes Softwareprodukt integriert werden. Dieses Softwareprodukt entsteht und wirkt durch die Integration der einzelnen Software-Komponenten. Die Komponenten wiederum sind entweder Open Source Eigen- oder Fremdentwicklungen oder proprietäre Eigen- oder Fremdentwicklungen [Rie11].
Um nicht bei jeder Entwicklung einer Komponente die Neu-Erfindung des Rades voran zu treiben, ist der Anteil der zugekauften beziehungsweise lizenzierten Fremdsoftware sehr hoch [Voa98]. Aus dieser Beobachtung leitet sich das Konzept der Softwarelieferkette ab. Aus dem Bereich des physikalischen Produktes, bestehend aus mechanischen Gütern, ist das Konzept der Warenlieferkette bereits seit langem bekannt. Als typisches Beispiel sei hier das Auto genannt. Ein Auto entsteht nicht durch die Fabrikation verschiedenster Teile in ein und dem selben Werk, von ein und dem selben Autobauer, sondern die einzelnen Teile werden von unterschiedlichen Spezialisten gefertigt und in einem Werk des Autobauers zum fertigen Endprodukt zusammengefügt. Dieses Konzept lässt sich auch auf die Produktion von Software anwenden.
Diese Lieferkette kann als rekursives Konstrukt verstanden werden, denn eine Komponente eines Zulieferers kann wieder nur ein Endprodukt einer zuvor durchlaufenen Lieferkette sein. Dass daraus in der global vernetzten Welt ein komplexes und kaum zu durchschauendes Netz an Zulieferern entsteht ist in der Realität nicht selten. Der Wettbewerb auf diesem globalen Markt wächst genauso wie der generelle Bedarf an flächendeckenden Softwarelösungen [Jon94].

1.1 Problemdefinition

Diese Rekursivität, Größe und Komplexität der kompletten Lieferkette eines Softwareproduktes bedingt ein hohes Maß an Vertrauen und Kooperation der Zulieferer und des Endproduktentwicklers, um gleichbleibend hohe Qualität des Endproduktes sicherzustellen [Hec99]. Viele Softwarefirmen können sich jedoch nicht auf das pure Vertrauen verlassen, denn mit dem Verkauf und der Nutzung von Software/-komponenten sind hohe Risiken verbunden. Es kann zu wirtschaftlichen, gesellschaftlichen und rechtlichen Schäden kommen, die durch Kontrolle anstelle puren Vertrauens vermieden werden sollen. Die Kontrolle der einzelnen Zulieferer kann durch Messen und Beurteilen der ausgeführten und definierten Prozesse geschehen. Dabei werden die Prozesse und Prozessdefinitionen im Zulieferunternehmen betrachtet und mit Hilfe eines standardisierten Maßstabs bewertet [CF02]. Diese Assessierung kann durch externe oder interne Fachleute durchgeführt werden.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, einen Vorschlag dieser Assessierung zu geben und ein Modell zur Zertifizierung aufzustellen. Diese Zertifizierung soll die Bewertung von Open Source Prozessen und Praktiken innerhalb einer Software Lieferkette definieren. Zusätzlich sollen die Risiken, die sich durch die Integration einer Open Source basierten Softwarelösung in das Softwareprodukt bieten, vermindert beziehungsweise gänzlich beseitigt werden. Das Beseitigen von Risiken kann durch das Implementieren von Standards beziehungsweise best practices voran getrieben werden. Diese best practice Methoden finden ihren Ursprung in Firmen und Beratungsunternehmen, welche die Open Source Entwicklungsprozesse umfassend und mit großer Sorgfalt umsetzen und pflegen.
Ein ganzheitliches Zertifizierungsmodell für Open Source Prozesse in der Lieferkette soll eine Differenzierung im Grad der Erfüllung bestimmter Ziele ermöglichen. Dies bedeutet, dass die assessierten Unternehmen in priorisierte Klassifikationsstufen eingeteilt werden, die die Qualität der geführten Open Source Prozesse zum Ausdruck bringen. Die Folgen für die an der Lieferkette beteiligten Entwicklungsfirmen sind:

- Ermöglichung einer transparenten Auswahl der Zulieferer an Hand des Ergebnisses der Zertifizierung
- Offenlegung interner Open Source Richtlinien

- Vermeidung von sonst nicht abschätzbaren Risiken
- Sicherstellung höchstmöglicher Qualität des Endproduktes und daraus resultierender Kundenzufriedenheit
- Objektivität durch externe Beurteilung

Das Aufstellen eines Zertifizierungsmodells speziell für Open Source Entwicklungsprozesse betont die unterschiedlichen Schwerpunktsetzungen in der Entwicklung von Software im Gegensatz zur proprietären Softwareentwicklung mit den proprietären Entwicklungsprozessen.
Zur Veranschaulichung des Prozesses wird in der Arbeit ein durchgängiges Fallbeispiel definiert. Darin werden unterschiedliche Szenarien durchlaufen, die an reale Situationen angelehnt sind.

1.3 Abgrenzung

Eine Abgrenzung erfolgt in der Beurteilung von Entwicklungsprozessen und nicht in der Beurteilung von Software Komponenten [MLP+01], [Wal04]. Dabei werden nicht Prozesse, um ein Endprodukt zu erhalten, bewertet, sondern die fertigen Komponenten oder Produkte. Diese Art der Bewertung lässt sich jedoch mit dem Konzept Open Source nicht vereinen, da dort das Prädikat *Entwicklung beendet* auf Grund der dynamischen Entwicklung schwer vergeben werden kann [FFM11].
Des Weiteren grenzt sich die vorliegende Arbeit auch von rein Community-basierten Open Source Software Entwicklungsprojekten ab. Die dort vorherrschenden organisatorischen, infrastrukturellen und ökonomischen Gegebenheiten sind nicht geeignet, um dort eine standardisierte Zertifizierung durchzuführen, die vergleichbar mit den Gegebenheiten von kommerziellen Software Lieferungketten ist [Ray98].
Zusätzlich wird diese Arbeit keine Betrachtung von Sicherheitsaspekten durchführen, da Sicherheitsaspekte für jede Art von Software wichtig und deshalb zu allgemein sind. Für diese Art der Zertifizierung sei auf andere Literatur verwiesen [HR06].

2 Methodik und Aufbau der Arbeit

In diesem Kapitel wird in Abschnitt 2.1 beschrieben, mit welcher Methodik die vorliegende Arbeit erstellt wurde. In Abschnitt 2.2 wird der weitere Aufbau der Arbeit skizziert. Abschließend wird in Sektion 2.3 das durchgängige Fallbeispiel eingeführt.

2.1 Methodik

Diese Arbeit wurde mit Hilfe einer umfangreichen Literaturrecherche erarbeitet. Als erster Schritt wurden die Grundlagen der Open Source Software erarbeitet, indem Primärliteratur zu diesem Thema gesucht wurde. Der Suchstring dazu war sehr simpel und bestand im Wesentlichen aus dem String *Open Source Software*. Nachdem das Fachgebiet ganzheitlich erarbeitet war, wurde nach den für diese Arbeit relevanten Spezialgebieten recherchiert. Diese lassen sich grob in drei Teile einteilen:

- Softwarelieferkette
- Softwareprozesszertifizierungsmodelle
- Open Source Einsatz in Unternehmen

In diesen drei Spezialgebieten wurde mit Hilfe von komplexen Suchstrings in Online Datenbanken nach Primärliteratur gesucht. In vielversprechender Primärliteratur wurde daraufhin Sekundärliteratur gesucht und in diese Arbeit mit einbezogen.
Da zu dem Thema Open Source sehr viel Literatur vorhanden ist, musste ein klarer und enger Filter angelegt werden, um nicht den Fokus auf die Aufgabenstellung zu verlieren. Daher ist es wichtig anzumerken, dass in Hinblick auf das breite Feld der Open Source Entwicklung kein Anspruch auf Vollständigkeit der Betrachtung erhoben wird. Insbesondere der wichtige Teilaspekt der Communitypflege wird in den folgenden Ausführungen nur am Rande betrachtet.

2.2 Aufbau der Arbeit

Im nächsten Kapitel (Kapitel 3) der Arbeit werden die Grundlagen dargestellt. Dort werden im Abschnitt 3.1 die allgemeinen Grundlagen der Entwicklung von Open Source Software erarbeitet, in Abschnitt 3.1.4 wird die spezielle Situation von Open Source Software in Unternehmen veranschaulicht, in Abschnitt 3.1.3 wird ein kurzer Überblick über geistiges Eigentum und den Zusammenhang mit Open Source Software gegeben, in Abschnitt 3.2 werden gängige Prozessbewertungsmodelle vorgestellt und in Abschnitt 3.3 wird die Softwarelieferkette beschrieben.
In Kapitel 4 werden Werke aus der Literatur vorgestellt, die einen ähnlichen Fokus haben wie die vorliegende Arbeit. Darauf werden in Kapitel 5 die Risiken der Benutzung von Open Source in Unternehmen dargestellt. Dies wird in den Abschnitten 5.2, 5.3 und 5.1 für ökonomische, gesellschaftliche und juristische Risiken weiter ausgeführt.
Das Kapitel 6 befasst sich mit den best Practices für den Einsatz von Open Source Software in den Unternehmen. Dafür wird in Abschnitt 6.1 die spezielle Softwarelieferkette mit Open Source Software vorgestellt. In Abschnitt 6.2 werden wichtige Merkmale und Eigenschaften einer Open Source Policy im Unternehmen aufgezeigt. Dies umfasst in dieser Arbeit neun Maßnahmen, die die Risiken aus Kapitel 5 reduzieren beziehungsweise vermindern sollen.
In Kapitel 7 wird ein Vorschlag für ein Open Source Zertifizierungsmodell vorgestellt. Dazu erfolgt in den Abschnitten 7.1, 7.2 und 7.4 die Motivation für das Modell, die Vorstellung des zugrundeliegenden Konzeptes und die Beschreibung des Prozesses. Daraufhin soll in Abschnitt 7.3 eine Klassifizierung in unterschiedliche Level gezeigt werden, woran sich eine Erläuterung zur Integration in die Lieferkette in Sektion 7.5 anschließt. Abschließend wird in Abschnitt 7.6 das Modell noch durch ein Fallbeispiel veranschaulicht, das in Abschnitt 2.3 definiert wird.
Abschließend werden im Kapitel 8 eine Zusammenfassung und ein Ausblick gegeben.

2.3 Definition eines Fallbeispiels

Das in diesem Abschnitt definierte Fallbeispiel soll den folgenden Teilen dieser Arbeit zur Veranschaulichung dienen. Das Beispiel orientiert sich an Firmen, die so oder in ähnlicher Form in der Praxis vorkommen können, aber rein fiktional sind. An dieser Stelle soll nur die grobe Charakteristik der Firmen angegeben werden, diese werden an

späterer Stelle im Text präzisiert und weiter beschrieben.
Für das Fallbeispiel werden drei Firmen definiert:

- PropForSuccess GmbH
- MixIT GmbH
- LibertatemVincIT GmbH

PropForSuccess GmbH ist eine Firma, die ganz auf proprietäre Software setzt und auch nur solche in ihrer Software benutzt. Im Gegensatz dazu ist LibertatemVinIT GmbH eine Firma, die voll und ganz auf Open Source Software setzt und auch nur solche Software in ihr Produkt einbaut. MixIT GmbH beschreitet einen Mittelweg aus beiden Extrempositionen und integriert sowohl Open Source Software als auch proprietären Code in ihr Produkt. Alle drei Firmen verkaufen ein Softwareprodukt an Endanwender, außerdem haben alle drei Firmen eine komplexe Lieferkette für ihre kommerzielle Softwarelösung.

3 Grundlagen

Dieses Kapitel beschreibt die Grundlagen, auf die in den darauf folgenden Kapiteln zurückgegriffen wird. Dazu gibt Sektion 3.1 eine Einführung in die Open Source Software Entwicklung. Anschließend erläutern die Teile 3.2 und 3.3 die Grundlagen der Softwareprozesszertifizierungsmodelle und der Softwarelieferkette.

3.1 Open Source Software

Die Open Source Softwareentwicklung ist kein neues Phänomen. Der Wunsch nach freier Software stammt bereits aus dem Jahre 1984. In diesem Jahr hat Richard Stallman die *freie Software Bewegung* gegründet, weil er keinen Zugriff auf den Source Code von Software hatte, die er zuvor selbst am Massachusetts Institute of Technology (MIT) mitentwickelte [FS05]. Stallmans Vision war es, dass jede Art von Software für alle offen und zugänglich ist [Fog05].
Ein weiterer Wegbereiter für die Open Source Bewegung war der finnische Student Linus Torvald. Er beschloss 1991 ein neues Betriebssystem zu entwickeln das kompatibel war zu dem kommerziellen und damals beliebten UNIX. Dieses Betriebssystem ist unter dem Namen Linux bekannt [V+05]. Linux ist ein Musterbeispiel für das kollaborative Open Source Entwicklungsmodell. Über das Internet ist es in der Theorie allen Menschen auf der Welt möglich ein Teil der Entwicklergemeinde zu werden.
Dies wird in dem bekannten Artikel von Eric Raymond *The cathedral and the bazaar* mit der Metapher *bazaar* treffend umschrieben, denn auf dem Marktplatz kann jeder Händler[1] seinen Beitrag leisten und verkaufen [Ray99a] Zusätzlich ist dieser Prozess von hoher Dynamik und weniger steifer Planung geprägt. Im Gegensatz dazu steht die klassische geschlossene Softwareentwicklung, die im Stile eines Kathedralenbaues[2] abläuft. Eine Kathedrale wird lange geplant und eben nach jenem Plan gebaut. Abweichungen

1 Dies symbolisiert hier den Entwickler
2 Dies ist ein Bild für ein komplexes und geschlossenes Softwareentwicklungsprojektes

vom Plan sind nicht leicht durchführbar, weshalb der Prozess eher als statisch beschrieben werden kann.
Ein wichtiger Grundsatz der Open Source[1] Bewegung ist: *free as in speech, not free as in beer* [Ken01]. Damit wird zum Ausdruck gebracht, dass die Bewegung ein liberaler Gegenpart zu der strikt geschlossenen und proprietären Entwicklung ist. Es bedeutet aber nicht, dass die Software kostenlos sein muss.

3.1.1 Die Open Source Definition

Zur Formalisierung der Bewegung hat sich die Open Source Definition entwickelt. Diese dient oft als Grundlage zur Bewertung, ob eine Software als Open Source Software eingeordnet werden kann oder nicht. Die einzelnen Punkte sind [P+99]:

- Freie Weiterverbreitung
- Zugang zum Source Code
- Erlaubnis zu abgeleiteten Arbeiten
- Sicherstellung der Integrität des ursprünglichen Codes
- Keine Diskriminierung gegen Personen oder Gruppen
- Keine Diskriminierung gegen einem Einsatzbereich
- Verbreitung der Lizenz
- Lizenz darf nicht produktspezifisch sein
- Lizenz darf keine andere Software befallen

3.1.2 Open Source Lizensierung

Es ist wichtig zu erwähnen, dass jedes Open Source Softwareprodukt unter einer Lizenz veröffentlicht werden muss. Diese Open Source Lizenz muss von der Open Source Initiative genehmigt werden, dass sie den Grundsätzen der Open Source Definition entspricht [Sch12]. Diese Lizenzen lassen sich grob in unterschiedliche Kategorien einordnen [KM01]:

- virale Lizenzen
- nicht-virale Lizenzen

1 Im Speziellen der Free Software Bewegung

Ein sehr weitverbreiteter Vertreter der viralen Lizenzen ist die GPL Lizenz[1]. Sie wurde von Richard Stallman ins Leben gerufen. Ein prominenter Vertreter für die nicht-viralen Lizenzen ist die Berkeley Software Distribution (BSD) Lizenz. Eine vollständige Liste der Lizenzen, die den Genehmigungsprozess überstanden haben, kann hier eingesehen werden [Ini13].

3.1.2.1 Virale Lizenzen

Als Beispiel für virale Lizenzen wird hier die GPL Lizenz erläutert. Grundsätzlich ist es erlaubt Software, die unter der GPL Lizenz veröffentlicht ist, zu kopieren, weiterzuverteilen und zu modifizieren [Fou07].
Sie beinhaltet den viralen Artikel 2b. Dieser besagt, dass jede Software, die GPL lizenzierte Software enthält, wieder unter der GPL Lizenz veröffentlicht werden muss [Pal03]. Dabei spielt der Begriff *abgeleitete Arbeit* eine große Rolle. Dieser steht unter Diskussion, da nicht eindeutig definiert ist, was genau abgeleitete Arbeit ist. Die Intention dieser viralen Komponente ist, dass jede Software, die unter der GPL Lizenz steht, für immer frei verfügbar bleibt und nicht einem Anderen vorenthalten werden kann. Der virale Teil wird oft auch als Copyleft bezeichnet. Darauf wird in Abschnitt 3.1.3 näher eingegangen. Genau dieser virale Teil ist es auch, der zu einer kontroversen Diskussion bezüglich der Einbindung von Open Source Software im Unternehmen führt (siehe Abschnitt 3.1.4). Dies wird als Risiko in Kapitel 5 näher beschrieben.

3.1.2.2 Nicht-virale Lizenzen

Ein Beispiel einer beliebten nicht-viralen Lizenz ist die BSD-Lizenz. Diese Lizenz fällt unter die Subkategorie der akademischen Lizenzen[2]. Weitere beliebte nicht-virale Lizenzen sind die Mozilla-Lizenz, die Artistic Lizenz und die Apache Lizenz [Ros04].
Die BSD-Lizenz macht keine Einschränkungen in der Benutzung der Software, einzig der Lizenz Text muss angezeigt werden [Ros04]. Diese Softwarekomponenten können sogar in proprietäre Softwareprodukte integriert werden. Ein Beispiel hierfür ist die Integration von BSD UNIX Code in das Mac OS X Betriebssystem von Apple [Lau08].

1 Zum Beispiel wird Software von der National Aeronautics and Space Administration (NASA) unter GPL veröffentlicht [Hah02]
2 Eine weitere beliebte Lizenz dieser Kategorie ist die MIT-Lizenz

3.1.3 Geistiges Eigentum

Wie im vorangegangen Abschnitt beschrieben, wird Open Source Software immer unter einer Lizenz veröffentlicht. Die Lizenzwahl trifft der ursprüngliche Autor der Software. Der ursprüngliche Autor ist auch der Besitzer dieser Software und besitzt daher ebenfalls das geistige Eigentum der Software [Ros04].
Der Besitz einer Kopie von Software bedeutet nicht, dass man auch das geistige Eigentum besitzt, sondern man ist Lizenznehmer und muss den Lizenzbedingungen entsprechen. Die Lizenzbedingungen schränken die Rechte, die man ausüben darf, ein. Geistiges Eigentum lässt sich in drei Bereiche einteilen (siehe Abbildung 3.1).

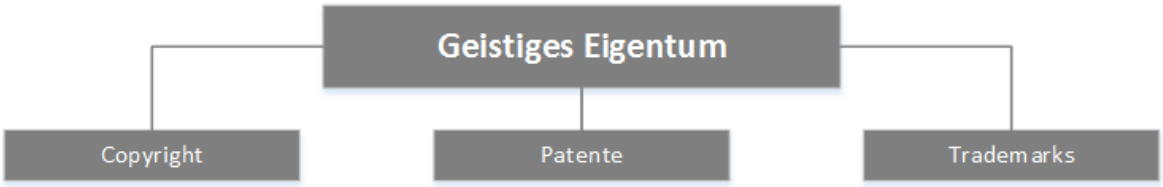


Bild 3.1: Veranschaulichung der verschiedenen Arten von geistigem Eigentum

3.1.3.1 Copyright

Jede neue Software ist durch Copyright-Rechte geschützt. Der Inhaber dieser Copyright Rechte ist der originäre Autor der Software. In der Open Source Softwareentwicklung ist jeder Entwickler, der einen Teil zu einem Softwareprojekt beiträgt, auch Besitzer des Copyrights. In Softwareentwicklungsfirmen besitzt die Firma immer das Copyright am Softwareprodukt. Wie in allen künstlerischen Bereichen schützt das Copyright kreative Ideen und ist auch ohne Beantragung und Registrierung auf der ganzen Welt gültig [Ros04]. Eine offizielle Anmeldung des Copyrights von Software ist erst seit 1980/81 möglich. Denn nur mit einer Registrierung kann man sein geistiges Eigentum vor Gericht geltend machen [Ros04].
Der in Abschnitt 3.1.2.1 genannte Begriff Copyleft ist eine weitverbreitete Anspielung auf das Copyright. Dies bedeutet im Gegensatz zu Copyright, dass Lizenznehmer die gleichen Rechte in Bezug auf die Verwendung von Software haben wie der Lizenzgeber - es darf keine Einschränkungen dieser Rechte geben. So wird jeder, der Software unter einer viralen/Copyleft Lizenz vertreibt, gezwungen die Weiterverbreitung unter der gleichen Lizenz vorzunehmen [FGL08].
Es kann, wie es zum Beispiel aus der Musikindustrie bekannt ist, zu Copyrightverletzun-

gen kommen, indem ein Fremder eine copyrightgeschützte Software kopiert. Dadurch kann es zu Klagen vor Gericht kommen, um die exklusiven Rechte, die man als Besitzer des geistigen Eigentums inne hat, durchzusetzen (näheres siehe Kapitel 5) [Ros04].

3.1.3.2 Patente

Im Gegensatz zu Copyright, das kreative Ideen schützt, schützen Patente technische und wissenschaftliche Verfahren und Ideen, die von unternehmerischer Relevanz sind [Ros04]. Patente müssen im jeweiligen Land beantragt werden und sind bei Genehmigung auch nur in diesem Land beziehungsweise in allen Ländern, die mit diesem Land ein Patentabkommen haben, gültig [Ros04].
Der Prozess zur Beantragung eines Patentes ist langwierig und teuer. Es gibt drei Kriterien die Patente erfüllen müssen, um genehmigt zu werden [ELF04]:

- Es muss eine Neuheit sein
- Es darf nicht offensichtlich sein
- Es darf nichts triviales sein

Software Patente sind zudem eine neue Erscheinung. Erst ab 1994 durften in den United States of America (USA) Software Patente ausgestellt werden. Vor 1970 waren Software-Bestandteile in keinster Weise patentierbar, weil sie wie mathematische Algorithmen behandelt wurden. Und ab 1981 durften Software Patente in Verbindung mit Hardware patentiert werden [ELF04].
Ein Problem mit Patenten und Open Source ist, dass viele Open Source Lizenzen, wie auch die GPL im Lizenztext, Patente vollkommen außen vor lassen. Dies hat viele Patentverletzungen zur Folge [Ros04]. Ein weiteres Problem mit Patenten ist, dass heutzutage sehr viele Patente vergeben werden und es damit zu einer Ansammlung von trivialen Patenten kommt [ELF04]. Weitere Probleme werden im Kapitel 5 aufgegriffen.

3.1.3.3 Trademark

Trademarks können Logos, spezielle Schriftzüge und Markennamen sein. Trademarks sind wichtig für Open Source Software Anbieter. Trademarks sind ein Alleinstellungsmerkmal für eine Software. Die Intention eines jeden Softwareentwicklers ist das Produzieren von qualitativ hervorragender Software, die jede Kundenbedürfnisse erfüllen. Anschließend verbindet jeder Käufer diese hohe Qualität mit der Trademark des Unternehmens, was

potentiell in weiteren Käufen mündet. Zusätzlich wird das Empfehlen dieser ausgezeichneten Produkte positiv beeinflusst [Ros04]. Alleinstellungsmerkmal deswegen, weil in den üblichen Open Source Lizenzen niemals Trademarks mit lizenziert werden. Diese werden immer vom ursprünglichen Open Source Software Entwickler gehalten [Ros04].

3.1.4 Open Source im Unternehmen

Der Begriff Trademarks führt zu einer weiteren Entwicklung im Bereich Open Source Software Entwicklung. Denn im Gegensatz zu den Anfangszeiten der Open Source Bewegung bildeten sich in der Folgezeit Gesellschaften und Unternehmen, die sich der Open Source Entwicklung verschrieben haben [Sch12].
Aus den vorher rein Community-orientierten und lose gekoppelten Entwicklerverbänden, formierten sich gemeinnützige und nicht gewinnorientierte Organisationen, um eine kontrollierte und gegenüber proprietären Softwareentwicklungsunternehmen besser aufgestellte Position inne zu haben [LT02]. Ein Beispiel hierfür ist die Apache Foundation. Zusätzlich veröffentlichten immer mehr vormals proprietäre Softwareentwickler wie Netscape Teile ihres Source Codes und stellten ihn als Open Source Projekt der Öffentlichkeit vor[1] [Ray99b], [LT02]. Andere große Softwarefirmen, wie zum Beispiel IBM, begannen Open Source Software einzusetzen und zu unterstützen[2] [DM05].
Zusätzlich bildeten sich völlig neue Unternehmen, die ihre Geschäftsstrategie auf die Entwicklung von Open Source Software ausgerichtet haben [Per05]. Diese Unternehmen produzieren kommerzielle Open Source Software und sind meistens der alleinige Entwickler des Software Produktes [Rie09]. Viele dieser Firmen fungieren als Service-orientierte Software Firmen, die ihren Profit aus Implementierungsservice oder Trainings- und Beratungsservice beziehen [Rie07]. Weiterer Einnahmekanal für Softwareentwickler ist das Dual-Licensing, bei dem die Software sowohl unter einer viralen Lizenz als auch unter einer freien Open Source Lizenz veröffentlicht wird. Für die Benutzung der Software unter einer freien Lizenz muss ein Entgelt entrichtet werden, denn nur damit lässt sich die virale Komponente verhindern und die kommerzielle Nutzung sicherstellen [Lin12].
Zusätzlich ist es möglich, Open Source Software unter dem Freemium Modell zu vertreiben. Dabei wird die grundlegende Software frei unter Open Source Lizenz vertrieben, jedoch fehlen dieser Version viele Merkmale die durch den Kauf der proprietären Version

1 In diesem Fall unter der gemeinnützigen Mozilla Foundation und den damit verbundenen Produkten
2 Sowohl mit Geld als auch mit Softwareprogrammierung

hinzugekauft werden können [Rie12]. Andere Einnahmequellen entstehen durch die Kopplung von Software mit Hardware [Ray99b].
Generell lässt sich festhalten, dass Open Source Software in der kommerziellen Welt der Software Entwicklung weit verbreitet ist und 82% aller Unternehmen Open Source Software einsetzen [BB02]. Zusätzlich wird das Open Source Entwicklungsparadigma auch innerhalb von Unternehmen angewandt. Dies wird Corporate Source genannt. Dabei betreibt ein Unternehmen einen Softwarepool an dem, mit Hilfe des Open Source Entwicklungsmodells, ohne Risiken gearbeitet werden kann [DG01].

3.2 Softwareprozesszertifizierungsmodelle

Da Software aus der heutigen Welt nicht mehr wegzudenken ist, wird man auch überall damit konfrontiert. Damit geht einher, dass diese Software einwandfrei funktionieren sollte - also die intendierte Funktionalität und Qualität mit sich bringt. Doch wie lässt sich diese Qualität messen und definieren?
Dazu bedarf es Standards oder best practices, die definieren was richtig, was falsch, was qualitativ gut oder was qualitativ schlecht ist [Tri02], [EvS00], [KKS12], [Fre01]. Diese Standards dienen sowohl als Grundlage zur Kommunikation als auch als Grundlage der Messung der Qualität der Software [Tri02]. Ein Sprichwort dazu lautet: "Guter Entwicklungsprozess ist, wenn der Benutzer zufrieden ist."[CF02].
Zertifizierung lässt sich damit definieren als Methode zur Beschreibung der Qualität und der Produktivität einer Software beziehungsweise eines Softwareentwicklungsprozesses. Zertifizierung kann als Benchmark oder Garantie und damit als Vergleichsgrundlage verschiedener Softwareeinheiten verstanden werden [Tri02], [Wal04]. Darüber hinaus ist es jedoch auch wichtig nicht nur die Produkt- beziehungsweise die Prozessqualität zu betrachten, sondern auch den sozialen Kontext, der in einer Softwareentwicklung vorherrscht, zu betrachten [CF02].
Zertifizierung kennt verschiedene Dimensionen [Wal04]:

- deskriptive versus normative Zertifizierung
- objektive versus subjektive Zertifizierung
- produktbasierte versus prozessbasierte Zertifizierung
- formale versus empirische Zertifizierung
- prozedurale versus mechanische Zertifizierung

- interne versus externe Zertifizierung

Die vorliegende Arbeit wird nur die prozessbasierte Zertifizierung behandeln. Dabei werden die Entwicklungsprozesse der Software betrachtet und assessiert.
In der proprietären Softwareentwicklung haben sich hierzu verschiedene Standards etabliert [O'H00]:

- CMM (Capability Maturity Model)
- ISO 9001
- SPICE

3.2.1 ISO 9001

In den späten achtziger Jahren wurde von der internationalen Standardisierungs Organisation die ISO 9001 Norm veröffentlicht. Sie deckt den vollständigen Produktlebenszyklus ab. Dieser beinhaltet das Design, die Entwicklung, die Produktion, die Installation und anschließende Serviceleistungen [BK96].
Diese Norm ist für eine Vielzahl von Produkten ausgelegt, also nicht nur für Software, sondern auch für Hardware. Die spezielle Norm ISO 9000-3 ist nur für die Bewertung von Software ausgelegt. Zur Zertifizierung werden Prozesse aus dem Unternehmen von einer externen Zertifizierungsstelle begutachtet und mit den Anforderungen aus dem ISO 9001 Standard verglichen. Anschließend werden die Ergebnisse bewertet. Heutzutage ist eine Zertifizierung nach ISO 9001 notwendig, um an großen internationalen Geschäften teilzunehmen [BK96].

3.2.2 CMM

Das Capability Maturity Model wurde aus einer Forschungsarbeit für das US-Verteidigungsministerium erarbeitet [BK96]. Es sollte die Qualität der Softwareentwicklung erhöht werden. Das ursprüngliche Ziel des CMM-Modells war es die Zulieferer in einer Softwarelieferkette zu bewerten, was jedoch später aufgeweicht wurde. Weiterhin wurden auch eigene Entwicklungsprozesse assessiert [BK96].
Die Prozesse im Unternehmen werden mit den Prozessen im Modell verglichen und bewertet und anschließend in die Maturity Level eingeteilt. Diese geben Aufschluss über den Grad der Erfüllung der Standards (je höher desto besser):

1 Initial

2 Repeatable

3 Defined

4 Managed

5 Optimising

3.2.3 SPICE

SPICE ist ein Softwarezertifizierungsstandard, der ins Leben gerufen wurde, um unterschiedliche lokale Entwicklungen von Zertifizierungsmodellen zu vereinheitlichen [BK96]. SPICE beurteilt die Prozesse des Unternehmens und vergleicht sie mit den Vorgaben, die der SPICE Standard definiert. Darauf aufbauend soll SPICE sowohl zur Determination der jetzigen Mächtigkeit der Prozesse herangezogen werden, als auch zur Definition von Prozessverbesserungen [BK96]. Heutzutage ist der SPICE Standard sehr weit verbreitet und es ist das Ziel dieses Modells sich, über die Branchen hinweg, als Standard-Zertifizierungsmodell zu etablieren.

Zusätzlich zu diesen Ansätzen bestehen noch Ansätze darin, eine Zertifizierung durch Benutzung durch den Käufer zu etablieren. Bei diesem Ansatz soll die Zertifizierung aufgrund der Zufriedenheit und des Extremtests der Benutzer vergeben werden [MLP+01], [SCA10].

3.3 Softwarelieferkette

Am Anfang der Softwareentwicklung bestand oft ein direkter Kontakt zwischen Entwickler und Benutzer der Software, wenn der Entwickler nicht sogar selbst der Benutzer war. Da Softwareprodukte heutzutage jedoch sehr komplex sind, ist dieses Verhältnis auch einer komplexeren Hierarchie von Agenten in einem Netzwerk gewichen [FF99]. Diese Hierarchie von Agenten ersetzt den Entwickler in der antiquierten Sicht auf die Softwareentwicklung und nennt sich Softwarelieferkette. Bis das fertige Softwareprodukt zum intendierten Benutzer kommt, kann die Software viele Stufen von Zulieferern durchlaufen. Am Ende der Lieferkette befindet sich meistens ein Softwareunternehmen, dass die einzelnen Fremdentwicklungen in die Eigenentwicklung integriert und dem Benutzer zur Verfügung

stellt.
In der Gegenwart kann nahezu kein Softwareentwickler ein komplexes Softwaresystem von Grund auf selbst schreiben. Er braucht dies auch nicht tun, da es für viele Funktionen die in einem Softwareprodukt stecken bereits passende Komponenten gibt [Voa00]. Generell stellt sich beim Design einer Applikation und der damit zu implementierenden Funktionalität die Frage nach Selbstentwicklung oder Integration einer Fremdapplikation[1] [FF99]. Dabei spielen verschiedene Aspekte eine Rolle:

- Kann man mit firmeneigener Expertise die gleiche Funktionalität herstellen wie ein spezialisierter Zulieferer [FF99]?
- Wie verlässlich und flexibel ist der Zulieferer [GDE97]?
- Wie gut ist die Unterstützung durch den Zulieferer [GDE97]?
- Wie sehr teilt man interne Informationen mit Zulieferern, was die Zusammenarbeit verbessern würde [Ver04]?
- Wie langlebig und flexibel ist die Fremdkomponente [Voa98]?
- Wie kann ich mit das NotInventedHere[2] Syndrom umgehen [Voa98]?
- Wie hoch sind die Kosten im Vergleich zur Eigenentwicklung? Beachte ich die Total Cost of Ownership[3] [HPL05]?
- Wieviel Anpassung der Komponente ist notwendig [NRS96]?
- Wie ist das generelle Ansehen des Zulieferers?

Aus diesen Fragestellungen muss die Softwareentwicklungsfirma die Entscheidung treffen, ob eine fremdentwickelte Software oder Softwarekomponente die Wahl ist oder ob man diese lieber selber entwickelt. Dabei gibt es heute nahezu keinen Entwickler, der die komplette Software selbst entwickelt. Ein Grund dafür ist unter anderem, dass der Wettbewerb in der globalisierten Welt immer weiter zunimmt [Jon94]. Weitere Gründe dafür sind, dass man nicht bei jeder Entwicklung das Rad neu erfinden muss, dass es oft billiger ist, auf Fremdentwicklungen zurückzugreifen, weil es schneller ist, als selbst zu entwickeln (time-to-market). Zusätzlich werden Fremdentwicklungen oft von Spezialisten

1 Auch make or buy Entscheidung genannt
2 Wenn die Software von einem Fremdentwickler kommt ist es für das eigene Personal oft schwer das volle Verständnis für die fremde Applikation zu erlernen
3 Dies sind nicht nur reine kosten zum Kauf der Komponenten, sondern die kompletten Kosten zum Kauf, Betrieb und Pflege der Software durch den Softwareproduktlebenszyklus

auf ihren Gebieten entwickelt [Voa98]. Es gibt dafür mehrere Ansätze Software zu entwickeln [Bre04]:

- Commercial of the Shelf - Enge Verbindung mit wenigen Zulieferern
- Component-based software engineering - Viele Zulieferer mit unterschiedlicher Beziehung
- Software service engineering - Web Service basierte Entwicklung

Welche Rolle Open Source Software in der Lieferkette spielt, wird in Abschnitt 6.1 näher betrachtet.

3.4 Zusammenfassung

Dieses Kapitel hat eine Einführung in die verschiedenen Problemfelder der vorliegenden Arbeit gegeben. Dazu hat der Abschnitt 3.1 die Grundlagen der Open Source Softwareentwicklung dargestellt. Dabei wurde, unter Berücksichtigung der Open Source Definition, die verschiedenen Lizenzierungsmodelle erläutert. Besonders wurde hierbei eine Unterscheidung der beiden Lizenzierungspole viral versus non-viral angegeben. Daran schloss sich eine kurze Beschreibung von geistigem Eigentum an, speziell wurde dabei auf die Begriffe Copyright, Patente und Trademarks eingegangen. Abschließend wurde in diesem Abschnitt dargelegt, wie Open Source im Unternehmen gebraucht wird und belegt, dass viele Unternehmen Open Source Software als Business Strategie verfolgen.
Im nächsten Abschnitt (3.2) dieses Kapitels wurde eine Einführung in verschiedene Softwareprozesszertifirierungsmodelle angegeben. Dabei wurde besonders auf das CMM, ISO 9001 und SPICE Framework eingegangen.
Der abschließende Teil (3.3) des Kapitels beschäftigte sich mit der Softwarelieferkette und der grundsätzlichen Problematik, warum und ob ein Unternehmen auf Software/-komponenten von Zulieferern zurückgreifen sollte oder nicht.

4 Verwandte Arbeiten

Dieses Kapitel stellt einige Arbeiten vor, deren Inhalt ähnlich zu der Vorliegenden ist. Es sei jedoch darauf hingewiesen, dass bei diesen Arbeiten der Fokus auf anderen Themenbereichen liegt. Im ersten Text in Abschnitt 4.1 wird ein Modell zur Zertifizierung von OpenLogic, einer nicht-wissenschaftlichen Firma, vorgestellt. In Sektion 4.2 wird ein Text über die Best Practices von Open Source Software vorgestellt. Abschnitte 4.3 und 4.4 behandeln jeweils praktische Anwendungsgebiete für eine Open Source Software Zertifizierung. Ersterer mit dem Fokus auf Android Apps, der Zweite ein Web Portal.

4.1 OpenLogic Certification Process for Open Source Software [Ope08]

In diesem Text aus dem Jahr 2008 stellt die Firma OpenLogic einen Prozess vor, mit dem Open Source Software Komponenten, die in ein Softwareentwicklungsprojekt integriert werden sollen, zertifiziert werden, bevor sie eingebaut werden. Die Intention des Prozesses ist es, dass ein zertifiziertes Produkt das Minimum an Integrität, Qualität und Unterstützbarkeit sicherstellt. Dazu werden vor allem folgende Kriterien überprüft:

- Kommt die Software aus einer guten Community?
- Ist das Lizenzmodell der Open Source Komponente verständlich?
- Werden grundlegende Qualitätsstandards in der Entwicklung eingehalten?
- Ist eine Dokumentation enthalten?

Als Werkzeuge, um die Integrität sicherzustellen, werden dem Unternehmen Code Scanner zur Verfügung gestellt. Diese werden dafür eingesetzt, dass jede einzubindende Komponente gescannt wird und damit auf kritische[1] Komponenten überprüft wird.

1 *kritisch* kann in diesem Zusammenhang zum Beispiel das Vorhandensein von GPL-Code sein.

Die Kandidaten zur Zertifizierung können durch Entwickler vorgeschlagen werden. Sicherzustellen ist jedoch, dass jede Komponente, die integriert werden soll, zertifiziert ist. Schlussendlich läuft der Prozess in vier Schritten ab:

1. **Package Selection:** In dieser Stufe wird das zu zertifizierende Paket definiert. Es muss ausgeschlossen werden, dass Redundanz auftritt und ein Paket mehrfach zertifiziert wird. Der Vorschlag zur Zertifizierung kann intern aus dem Unternehmen selbst kommen oder von extern durch Markteinflüsse.
2. **Prequalification:** Nachdem ein Paket definiert ist, wird in diesem Schritt die Lizenz grundlegend überprüft. Weiterhin wird die Verfügbarkeit des Source Codes überprüft. Nur dann kann die nächste Prozessstufe erreicht werden.
3. **Initial Acceptance:** In dieser Stufe des Prozesses werden die Gegebenheiten in Bezug auf die Unterstützung des ursprünglichen Entwicklers getestet. Des Weiteren wird analysiert, wie die rechtliche Situation des Zulieferers und des eigenen Unternehmens ist. Es wird zusätzlich geprüft, ob die Komponente Sicherheitsprobleme hat und wie es um die Verteilungswege des Paketes bestellt ist.
4. **Full Certification:** In dieser Stufe werden die Schritte, die man in den vorherigen Stufen des Prozesses bereits begutachtet hat, noch einmal in detaillierter Form wiederholt. Es wird eine komplette und genaue Überprüfung des Lizenzmodells durchgeführt, welche die Dokumentation auf Verfügbarkeit, Vollständigkeit und Konsistenz testet und die Lieferwege verfolgt.

Wenn ein Paket die Stufe 4 des Prozesses durchlaufen hat, ist es zertifiziert und kann darauf im Unternehmen ohne Bedenken eingesetzt werden.

4.2 Open Source Standards on Software Process: A Practical Application [Fre01]

In diesem Paper werden Prozessstandards für Open Source Softwareentwicklung eingeführt. Für die Bewertung werden bislang immer Checklisten-ähnliche Formate eingesetzt. Dabei wird der Prozess im Unternehmen begutachtet und überprüft, ob er den Kriterien auf der Checkliste entspricht und, wenn ja, wie groß die Übereinstimmung ist. Dabei beruht die Überprüfung der Qualität der Übereinstimmung auf best practices und Standards. Ohne Standards wäre eine solche Qualitätsmessung nicht möglich, da keine

Vergleichsgrundlage zur Verfügung stehen würde.
Daraufhin werden in dem Text die aus Abschnitt 3.2 bekannten Modelle mit ihren definierten und zu Grunde liegenden Standards vorgestellt. Daraufhin wird versucht Standards für die Open Source Entwicklung einzuführen.
Dabei kristallisieren sich drei Hauptaktivitätsfelder heraus:

Administrative Aufgaben: Hiermit sind Aktivitäten gemeint, die dafür sorgen, dass alle Interessensgruppen zusammenarbeiten und kommunizieren können, um einen reibungslosen Prozessablauf sicherzustellen. Ein wichtiger Bestandteil ist das Konfigurationsmanagement.

Konstruktion von Support Material: Das Support Material ist wichtig, um das Verständnis für die Standards zu festigen und in vielen Einsatzbereichen anzunähern. Dazu können zum Beispiel neue E-Learning Plattformen eingesetzt werden.

Verfügbarer und lauffähiger Source Code: Dies ist der wichtigste Teil. Jeder Teil der Standards sollte in erkundbarer und nachvollziehbarer Fassung verfügbar sein, um einen Lernprozess anzustoßen.

Der Standard sollte sowohl in textueller, als auch in grafischer Form verfügbar gemacht werden.

4.3 A Certification Process for Android Applications [KKS12]

Grundlegend für diese Arbeit ist die Annahme, dass Android in den letzten Jahren eines der Zugpferde der Open Source Entwicklung wurde. Android ist mittlerweile das am weitesten verbreitete mobile Betriebssystem und ermöglicht es Entwicklern, Open Source Software zu teilen. Zusätzlich ist das Betriebssystem offen und ermöglicht eine Anpassung an eigene Vorstellungen und Wünsche.
Durch die rasante Entwicklung der Android Plattform in den letzten Jahren ist es jedoch schwierig, eine validierte und auf Sicherheitslücken überprüfte Fassung einer Applikation sicherzustellen. Die Intention des Textes ist es, diese Lücke zu füllen und eine Checklisten-basierte Validierungsmöglichkeit für mobile Applikationen vorzustellen.
Der Zertifizierungsprozess, der dafür vorgeschlagen wird, setzt sich aus verschiedenen bereits vorhandenen Quellen und Tools zusammen. Der Prozess soll unter allen App-Entwicklern ausgerollt werden, des Weiteren sollen App-Entwickler ihren Source Code

zur Verfügung stellen, um eine noch höhere Sicherheit zu erfüllen.
Der eigentliche Prozess wird noch nicht präsentiert, soll jedoch durch akademische Bemühungen schnell erstellt werden.

4.4 A Web Portal for Certification of Open Source Software [MFS12]

Dieser Text stellt ein Web Portal vor, das dazu konzipiert wurde Open Source Software zu zertifizieren. Es wird unterstellt und angenommen, dass heutzutage Open Source Software in vielen Softwareentwicklungsprojekten wiederverwendet und integriert wird. Ein Entwickler sucht demnach immer nach vorhanden Lösungen bevor er selbst tätig wird und das Implementieren beginnt. Genau dieser Entwickler möchte jedoch kein risikobehaftetes, beziehungsweise schlechtes Open Source Software Produkt in sein Produkt integrieren, weshalb eine Zertifizierung unumgänglich scheint.
In diesem Text wird der Begriff Zertifizierung für den Prozess verwendet: Analyse eines Source Code Dokumentes und der Erstellung eines Reports. Dabei wird unterschieden, ob das Open Source Produkt aus mehreren Source Code Dokumenten besteht oder nicht. Wenn ja, dann wird für jedes Source Code Dokument ein Report angefertigt, der aus den verschiedenen Teilen zusammengesetzt wird. Das Ganze geschieht Programmiersprachen-unabhängig. Dazu sind die Reports, die erstellt werden, für die Benutzer anpassbar.
Als weiteren wichtigen Punkt in dem Aufsatz wird eine Domänen-spezifische Sprache eingeführt, die den Zertifizierungsprozess anpassbar macht und neue Analysetools erweiterbar werden.

5 Risiken

In diesem Kapitel werden die Risiken der Benutzung von Open Source Software im Rahmen einer Softwarelieferkette aufgeführt. Diese Risiken können vielfältig sein und lassen sich grob in drei Risiken einteilen, die zu jeweils unterschiedlichen Folgen führen können(siehe Abbildung 5.1).
In Abschnitt 5.1 werden die juristischen Risiken betrachtet. In Abschnitt 5.2 werden die wirtschaftlichen Risiken und Folgen durch die Einbindung betrachtet. In der darauffolgenden Sektion 5.3 werden die gesellschaftlichen Folgen betrachtet. Abschließend wird in Abschnitt 5.5 eine Zusammenfassung des Kapitels dargestellt.
Es sei darauf hingewiesen, dass die einzelnen Bereiche keine klaren Grenzen aufweisen und es durchaus zu Überschneidungen kommen kann. Dies ist offensichtlich, da juristische Folgen oft auch wirtschaftliche und gesellschaftliche Folgen haben können (siehe Abbildung 5.1). Außerdem wird versucht in diesem Kapitel das Fallbeispiel aus Abschnitt 2.3 aufzugreifen und weiter auszuführen.

Bild 5.1: Veranschaulichung der verschiedenen Risikoarten

5.1 Juristische Risiken

Wie in Abschnitt 3.1.3 beschrieben, wird Software durch bestimmte Rechte bezüglich geistigem Eigentum geschützt. Im Speziellen wird auch Open Source Software durch diese Rechte geschützt. Jeder der durch kreative Ideen oder logische Erfindungen eine neuartige Software/-komponente erschafft, ist im Besitz dieser Rechte. Genauso wie in

der Musikindustrie lassen sich solche Rechte auch durchsetzen, denn die Benutzung von fremdem geistigen Eigentum ist verboten, wenn man keine Lizenz dafür besitzt.
In den nächsten Abschnitten werden einige Risiken für das Einbetten von Open Source Software im eigenen Softwareentwicklungsprojekt betrachtet. Es wird dabei kein Anspruch auf Vollständigkeit erhoben. Es stellt jedoch eine Auswahl an Beispielen dar, die für die meisten Einsatzgebiete ausreichend sind.

5.1.1 Probleme mit Lizenzen am Beispiel der GPL

Open Source Software wird immer zusammen mit einer Open Source Lizenz verteilt. Eine dieser Lizenzen ist, wie in Abschnitt 3.1.2.1 beschrieben, die GPL Lizenz. Dieser Lizenz liegt das Copyleft Prinzip zu Grunde. Jeder Softwareentwickler darf den GPL-lizenzierten Code nutzen, muss jedoch abgeleitete Arbeit auch unter der GPL veröffentlichen [Ros04]. Dies kann zum Beispiel Offenlegung von eigenem geistigen Eigentum zur Folge haben. Durch das Einhalten der Lizenz wird der Zugang zu allen GPL-lizenzierten Softwaremodulen sichergestellt. In der Lizenz ist festgehalten, dass man die Lizenzvereinbarungen akzeptiert, sobald man die Software benutzt [Fou07]. Wenn man gegen die Lizenzvereinbarungen verstößt, ist in der Lizenz geregelt, dass es zur sofortigen Beendigung der in der Lizenz aufgelisteten Rechte kommt. Das hat zur Folge, dass man die Software nicht mehr benutzen, verändern und weiter verteilen darf.
Bei Verletzung der GPL wird man von der Free Software Foundation aufgefordert die Benutzung sofort einzustellen oder den Code auch unter der GPL-Lizenz zu veröffentlichen. Des Weiteren kann es zu Gerichtsverfahren kommen [Ros04].
Bei der GPL ist der Hauptstreitpunkt der Begriff der *abgeleiteten Arbeit*. Viele Entwickler, die GPL-lizenzierte Software verändern oder verändert in die eigene Software einbauen, haben zu diesem Streitpunkt eine andere Sichtweise, als der Besitzer der GPL-lizenzierten Software. Das Problem ist, dass der Begriff der abgeleiteten Arbeit in der Lizenz sehr schwammig definiert ist und Raum für Interpretationen lässt [Fou07], [Ros04].
Nach [Ros04] ist das Nachweisen von abgeleiteter Arbeit sehr schwer, denn man muss auf der einen Seite das physikalische Kopieren nachweisen[1], auf der anderen Seite ist der Vergleich auf Ähnlichkeit schwierig, weil ein hohes Abstraktionsniveau notwendig ist. Es wird festgehalten, dass es meistens schneller ist die entsprechenden Teile der Software

1 Dies beinhaltet den physikalischen Zugang zur Quelle und die Eingabe des Kopierbefehls

selber zu schreiben als zu kopieren.
Als einen Ausweg aus der Situation bot die Free Software Foundation eine schwächer virale Lizenz an: die Lesser GNU General Public License (LGPL). Damit ist das Einbinden oder Linken von GPL-lizenzierten Bibliotheken ohne Risiken möglich [Ros04].

5.1.2 Probleme mit Copyright

Jeder Teil von Software kann potentiell Copyright enthalten. In der Open Source Softwarelizenz wird dieses Copyright auf den Lizenznehmer zur Benutzung übertragen. Der Besitzer des Copyrights bleibt aber der ursprüngliche Entwickler der Software. Der Lizenzgeber überträgt die Software, die er weiter verteilt, meistens mit der *as-is*-Klausel [Lau08]. Dazu kommt oft eine *no-warranty*-Klausel. Dies bedeutet, dass der Lizenzgeber keine Garantien für die Software übernimmt und für etwaige Copyrightverletzungen nicht einsteht. Dies würde dann zum Tragen kommen, wenn ein Lizenznehmer wegen einer Copyrightverletzung angeklagt wird, dann würde der Lizenzgeber keine Haftung für dem entstandenen Schaden übernehmen. Nach deutschem Recht sind solche Klauseln nicht rechtskräftig [MJ01].
Der Ankläger in diesem Fall muss sein Copyright registriert haben, andernfalls ist ein Durchsetzen seiner Rechte vor Gericht nicht möglich. Bis jetzt kam es zu keinen nennenswerten Copyrightverletzungsklagen/-verurteilungen [Ros04].

5.1.3 Probleme mit Patenten

Ein noch größeres Problem im Vergleich zu den Copyright bedingten Verletzungen ist das Problem der Patentverletzungen [HH11]. Pro Jahr gibt es demnach 200.000 neue Patentprobleme und 55 Gerichtsfälle pro Woche. Dies gilt auch im Open Source Umfeld. Ein Grund dafür ist, dass viele Open Source Lizenzen Patente außen vor lassen beziehungsweise implizite Patentlizenzen verteilen[1] [ELF04]. Erst die GPL Version 3 gibt alle Patente an die Lizenznehmer weiter. Dies schreckt jedoch viele Entwickler ab diese Version zu gebrauchen, denn man befürchtet seine Patente dadurch kostenlos offenzulegen, was zum Verlust des eigenen geistigem Eigentums führen würde [BFM07]. Durch dieses Fehlen der Patentregelungen in der Lizenz müsste der Lizenznehmer zu allen verwendeten Patenten eine Lizenz beim Patentbesitzer kaufen oder erstehen. Ein

1 Ein Beispiel hierfür ist die GPL Lizenz Version 1

Problem an dieser, auf den ersten Blick, einfachen Herangehensweise ist, dass es für kleine Entwickler praktisch unmöglich ist festzustellen, welche Patente in der benutzten Software enthalten sind [Pat13]. Ein weiteres Problem für kleine Entwickler ist, dass sie durch fehlendes Kapital auch wenig eigene Forschung betreiben können, um eigene Patente auf den Markt zu bringen [Pat13].
Schlussendlich kann es durch Patentverletzungen auch zu Gerichtsfällen kommen, in denen der Patentbesitzer den unrechtmäßigen Patentbenutzer verklagt. Die Forderungen können verschiedene Ausprägungen haben. Als Beispiele seien hier finanzielle Ausgleichszahlungen oder Einstellung der Verwendung des Patents genannt [Pat13], [Kos07].
Die Softwareindustrie hat sich auch stark verändert, was in größerer Komplexität resultiert. Es werden defensiv und offensiv Patente beantragt. Defensive Patente dienen zum Schutz, um im Falle einer eigenen Patentverletzung den Ankläger mit der Verletzung von eigenen Patenten anzuklagen [Pat13], [ELF04]. Offensive Patentbeantragung findet zum Beispiel von sogenannten Patent Trolls statt. Diese sind kommerziell nicht als Softwareentwickler tätig, sondern sie haben sich darauf spezialisiert andere Entwickler der Patentverletzung anzuklagen. Der Angeklagte in diesem Fall muss jedoch nur die Nicht-Tätigkeit nachweisen, um einer Verurteilung zu entgehen [Pat13]. In Abschnitt 5.1.4 werden einige Beispiele für solche *Patentkriege* aufgeführt.
Als Beispiel einer besonderen Art von Patentproblemen soll hier die **Patent-Retaliation-Klausel** betrachtet werden. Diese Art der Klausel ist zum Beispiel in der Mozilla Public License (MPL) 2.0 oder in der Apache 2.0 Lizenz enthalten. Der Ausgangspunkt für diese Klausel ist, dass ein Lizenzgeber einem Lizenznehmer ein Softwareprodukt mit einer Lizenz zur Verfügung stellt. Diese Lizenz enthält zudem die Retaliation Klausel. Der Lizenznehmer stellt nun fest, dass in der Software eigene Patente verletzt werden und will den Lizenzgeber der Patentverletzung anklagen. Durch die Retaliation-Klausel ist der Lizenzgeber daraufhin in der Lage die Lizenz der zuvor zur Verfügung gestellten Software sofort zu entziehen. Die Lizenz erlischt sofort [RE04a]. Die Sublizenznehmer des Lizenznehmers verlieren daraufhin auch die Lizenz.
Es gibt zwei Arten dieser Klausel:

- eine schwache
- eine starke

Die schwache Retaliation Klausel ist im Endeffekt äquivalent zu dem oben Beschriebenen, denn sie beinhaltet nur die lizenzierte Software des Lizenzgebers. Das heißt, nur wenn der Lizenzgeber vom Lizenznehmer mit Ziel der zwischen den beiden Parteien lizenzierten

Software der Patentverletzung bezichtigt wird, greift die schwache Retaliation Klausel [RE04a].
Im Gegensatz dazu greift die starke Retaliation Klausel bei jeder Art von Patentverletzungsanklage des Lizenznehmers an den Lizenzgeber. Dies schreckt viele Unternehmen ab Software mit einer starken Retaliation Klausel zu lizenzieren, da dadurch legal Patentverletzungen möglich sind [RE04a]. Ein Beispiel wird im Abschnitt 5.1.4 angegeben. Abbildung 5.2 zeigt die Voraussetzungen der Retaliation Klausel.

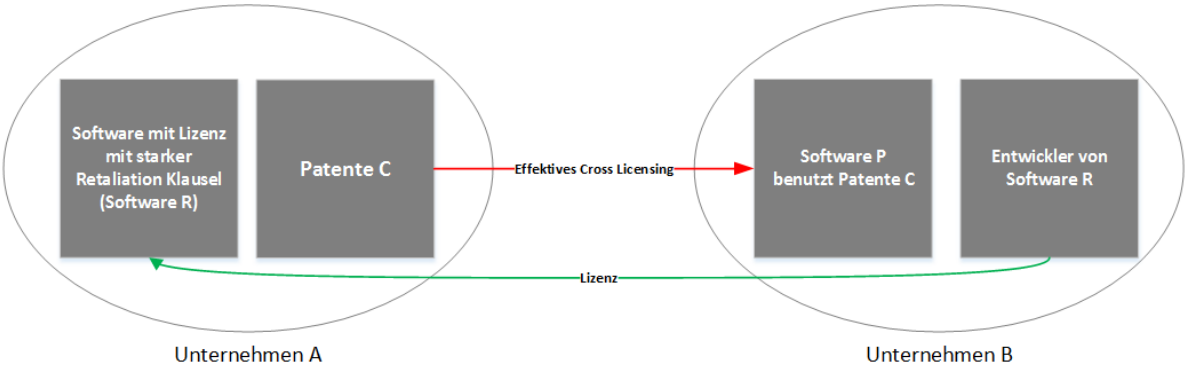


Bild 5.2: Veranschaulichung der Retaliation Klausel

5.1.4 Beispiele aus der Praxis

In den folgenden Sektionen sollen Beispiele für juristische Risiken aufgezeigt werden. Auf Grund der Vielzahl an Problemfällen werden nur Klagen/Fälle mit großer Medienwirksamkeit aufgeführt.

5.1.4.1 Beispiel für GPL-Verletzung

Als komplexes Beispiel für die juristischen Folgen von GPL-Verletzungen soll die Firma Cisco dienen. Cisco kaufte die Firma Linksys für ungefähr 500 Millionen Dollar auf. Linksys produziert kabellose Produkte. Teile dieser kabellosen Produkte werden von Broadcom, einem Halbleiter-Hersteller für Breitband-Kommunikationsmöglichkeiten, hergestellt. Treiber für diese Teile werden von einem Third Party Entwickler produziert. Diese Treiber wurden an Linksys lizensiert. Nach der Übernahme von Linksys stellte Cisco fest, dass in den Treibern GPL Code enthalten ist. Somit liegt eine GPL-Lizenzverletzung

vor und die Free Software Foundation versuchte Cisco dazu zu zwingen, den Source Code offenzulegen, was Cisco nach Monaten von Verhandlungen tat [FAK04].

5.1.4.2 Beispiele für Patentklagen

Als erstes Beispiel sei hier der Fall von Microsoft gegen TomTom aus dem Jahre 2008 genannt. In diesem versuchte Microsoft das Einführen von TomTom Navigationssystemen in die USA komplett zu unterbinden. Grund dafür war eine Patentverletzung von Microsofts File Allocation Table (FAT) Dateisystempatent. Das Verfahren wurde nach Zahlung von TomTom eingestellt [Pat13].
Als zweites Beispiel der Fall Apple gegen HTC Corporation aus dem Jahre 2010. In diesem Fall hat Apple HTC der Patentverletzung in 20 Fällen bezichtigt und wollte das Einführen von HTC Produkten in die USA komplett unterbinden. Am Beispiel von diesem Fall sollen zwei Verhaltensweisen bei Patentstreitigkeiten veranschaulicht werden:

- Das Benutzen von eigenen Patenten um eine Gegenanklage zu starten
- Oft sind die offensichtlichen Angeklagten nicht die wirklichen Angeklagten

In diesem Fall eröffnete HTC eine Gegenklage auf Grund von fünf Patentverletzungen und wollte damit den Verkauf von Apple Produkten unterbinden. Die zweite Besonderheit ist, dass das eigentliche Ziel wohl nicht HTC war, sondern Google's Betriebssystem Android und Google selbst, da die Patentverletzungen von HTC auf Grund von Android entstanden sind [Pat13].

5.1.4.3 Beispiel für effektiven und weitreichenden Einsatz von Retaliation Klausel

Als Beispiel für den Einsatz der starken Retaliation Klausel lässt sich die Firma Apple nennen. Damit nutzt jeder Lizenznehmer von Apple, Open Source Software unter der Bedingung, dass man Apple nicht auf Patentverletzung anklagt, da die Lizenz zu dem Apple Produkt ansonsten sofort erlischt. Apples Open Source Software hat sich in vielen Bereichen jedoch bereits unverzichtbar gemacht, was für Apple bedeutet, dass sie jedes Patent ihrer Lizenznehmer, ohne die Gefahr einer Patentklage, benutzen dürfen. Apple kann dieses Vorgehen daher als effektives und kostengünstiges Cross-Licensing Abkommen gegen andere Unternehmen nutzen [RE04a].

5.2 Ökonomische Risiken

Wie bereits in der Einleitung zu diesem Kapitel erläutert, hängen die Risiken, die sich durch die Einbindung von Open Source Software ergeben, oft von einander ab. So ergeben sich aus den im Abschnitt 5.1 angeführten Risiken auch häufig ökonomische Risiken. Dies soll in diesem Abschnitt erläutert werden.

5.2.1 Direkter ökonomischer Schaden

Als erstes ökonomisches Risiko lässt sich festhalten, dass mit jedem Gerichtsverfahren, dass gegen ein Unternehmen geführt wird, auch ein direkter finanzieller Schaden verbunden ist. Dies beinhaltet nicht nur die Gerichts- und Anwaltskosten, sondern auch die etwaigen Strafzahlungen [Hel01]. Wie im vorherigen Abschnitt beschrieben, kann es zu Gerichtsverhandlungen auf Grund von Patentverletzungen, Copyrightverletzungen oder Softwarelizenzverletzungen kommen. Diese Art des Schadens wird als direkter Schaden klassifiziert.

5.2.2 Indirekter ökonomischer Schaden

Was jedoch weitaus schlimmer ist, ist der indirekte Schaden, der durch juristische Folgen entsteht. Wie die in Abschnitt 5.1.4 dargestellten Beispiele zeigen, können solche Gerichtsverfahren weitreichende Folgen für ein Unternehmen haben. Wenn man annimmt, dass Apple in dem Fall gegen HTC als Gewinner aus dem Gerichtssaal hervorgeht, dann kann man sich auch ausmalen, was dies für HTC und deren Verkaufszahlen in den USA bedeuten würde. Um dies zu verhindern, wird wie im Falle von Microsoft gegen TomTom meist eine hohe Ausgleichszahlung geleistet, um dem Verkaufs-/Einfuhrstopp zu entgehen [Pat13]. In die gleiche Kategorie lassen sich auch die Probleme mit Lizenzen[1] einordnen, denn wenn die Lizenz für das lizenzierte Stück Software entzogen wird, dann resultiert das immer in einem finanziellen Schaden, wenn das Softwarestück von großer Bedeutung für das Geschäftsmodell oder das Softwareprodukt für ein Unternehmen ist. Dann ist es von Nöten aufwändiges Re-Engineering zu betreiben, um den Code umzuschreiben oder eine andere Softwarekomponente zu benutzen oder den Lizenzbedingen zu entsprechen, was

1 Retaliation Klausel und zum Beispiel GPL Lizenzprobleme

auch wieder zusätzliche Kosten verursacht. Das Entsprechen der Lizenzbedingungen hätte jedoch im Fall der GPL Lizenz zur Folge, dass der komplette Code unter Umständen unter der GPL Lizenz veröffentlicht werden müsste. Damit wäre der Code für jeden anderen Entwickler einsehbar - also auch für die Konkurrenten. Im schlimmsten Falle wäre damit das Geschäftsmodell eines ganzen Unternehmens zerstört [Man06], [Egg06].

5.2.3 Passiver ökonomischer Schaden

Als dritte Kategorie des ökonomischen Schadens - auf Grund von juristischen Problemen - gibt es den passiven finanziellen Schaden. Dieser besteht für jede Art von Softwareentwicklern, besonders jedoch für die kleinen und mittelgroßen Entwickler. Dabei handelt es sich um den Schaden der potentiell im Produktportfolio vorhanden ist. Jeder, der Open Source Software in seinem Entwicklungsprojekt einsetzt, muss damit rechnen, dass er Patente oder Copyright eines fremden Unternehmens verletzt. Für größere Softwareunternehmen mit großem eigenen Patentportfolio führt dies meist unweigerlich zu einem Patentkrieg. Für kleinere Unternehmen kann dies im schlimmsten Fall das Aus der Unternehmung bedeuten, da sie entweder nicht in der Lage sind den finanziellen Forderungen nachzukommen oder weil ihr Geschäftsmodell unmittelbar auf fremdem geistigen Eigentum fußt [Pat13].

5.2.4 Versicherungen im Falle einer Anklage wegen Verletzung von geistigem Eigentum

An dieser Stelle wird kurz auf das Modell der Versicherungen gegen Klagen auf Verletzung von geistigem Eigentum eingegangen. Es besteht die Möglichkeit sich gegen die finanziellen Forderungen, die auf Grund von Verletzungen von Patent- oder Copyrightrechten entstanden sind zu versichern. In einem etwaigen Fall würde die Versicherung den entstandenen Schaden begleichen. Dieses Konzept ist für Open Source Software nötig, weil in den Lizenzen keine Garantien übernommen werden, wie in Abschnitt 5.1 aufgeführt wurde [Egg06]. Kleinere Unternehmen werden von solchen Versicherungen jedoch auf Grund des hohen Risikos nicht berücksichtigt.

5.3 Gesellschaftliche Risiken

Laut [WL01] ist der größte Schaden, den man durch das falsche Benutzen von Open Source Software erleiden kann, der gesellschaftliche Schaden. Dieser kann mit ökonomischem Schaden verbunden sein, aber im Vordergrund steht die öffentliche Ächtung der betroffenen Unternehmen. Diese Ächtung erfolgt auf Grund der Unterwanderung der Open Source Prinzipien. Ein Beispiel dafür wäre das Einbinden von Open Source Code in ein wirtschaftlich sehr erfolgreiches Produkt, aber keiner Nennung oder Danksagung an die Entwickler dieser Open Source Software.
Eine mögliche Folge der Ächtung wäre, dass das ehemals erfolgreiche Produkt nicht mehr gekauft werden würde. Dies würde für die betroffene Firma wieder finanzielle Einbußen bedeuten, im Vordergrund würde jedoch der gesellschaftliche Faktor stehen. Das Image und die Reputation können das Käuferverhalten maßgeblich beeinflussen und jede Firma müsste versuchen diesen Reputationsverlust wieder zu beseitigen [WL01], [Hel01].
Als Beispiel hierfür seien die Firmen nVidia und Microsoft genannt. nVidia passte den XFree86 Grafikkartentreiber für ihre eigenen Treiber an, aber veröffentlichte trotz GPL Code den Source Code nicht. Daraufhin sah sich nVidia gezwungen den Code komplett zu entfernen und die Treiber ohne Lizenzverletzungen wieder zu veröffentlichen, da die Öffentlichkeit auf das Verhalten schlecht reagierte. Bei dem Microsoft Fall kaufte Microsoft die Firma Softway Systems, die GPL basierte Software veröffentlichte, und verpackte die Software von Softway Systems unter dem Namen Microsoft Interix. Dies führte zu einer Ächtung dieses Produktes [WL01].

5.4 Veranschaulichung am Fallbeispiel

An dieser Stelle wird das in Abschnitt 2.3 eingeführte Beispiel aufgegriffen und an Hand der Beispielunternehmen erläutert, wie sich die Risiken im Unternehmen zeigen. Für jedes Unternehmen wird ein beispielhaftes Szenario angegeben, das im nachfolgenden Kapitel fortgesetzt wird.

5.4.1 Beispielszenario 1: PropForSuccess GmbH

Das Unternehmen PropForSuccess GmbH bedient sich für seine Softwarelösung aus einem Softwarekomponenten-Pool, der nur aus proprietären Lösungen besteht. Das Problem, das für diese Firma entstehen kann, ist, dass sich für eine der Komponenten herausstellt, dass in dieser GPL lizenzierter Code vorhanden ist, beziehungsweise eine der Komponenten eine abgeleitete Arbeit einer GPL-lizenzierten Arbeit ist. Dies kann für PropForSuccess dazu führen, dass sie aufgefordert werden ihre Softwarelösung nun auch unter der GPL Lizenz zu veröffentlichen. Da die GPL-lizenzierte Softwarekomponente auch noch ein integraler Bestandteil des Softwareproduktes ist, steht PropForSuccess vor einer schwierigen und weitreichenden Entscheidung: Entweder man entschließt sich den Open Source Weg zu gehen oder man sieht sich nach Alternativen um. Die Folgen sind sowohl mit hohem Aufwand und finanzieller Umorientierung oder monetären Kosten verbunden. In jedem Fall hat die Unternehmensführung erkannt, dass Open Source Software in allen Teilen der Softwareindustrie eine Rolle spielt und zusätzlich, dass Vertrauen in einen Zulieferer einer Softwarekomponente wichtig, aber nicht das Entscheidende ist. Vielmehr muss die Kontrolle eine größere Rolle spielen, sonst werden weiterhin große Risiken für die Unternehmung eingegangen.

5.4.2 Beispielszenario 2: MixIT GmbH

Anders als PropForSuccess GmbH bezieht MixIT ihre Softwarekomponenten sowohl von Open Source Zulieferern, als auch von proprietären Entwicklern. Die MixIT GmbH ist spezialisiert auf die Entwicklung von Entwicklungsumgebungen und Bildbearbeitungsprogrammen für das Windows Betriebssystem und Mac OS. Für die Entwicklung einer neuen Version ihres Bildbearbeitungsprogramms war es nötig, eine Komponente, die unter der Apple Open Source Lizenz veröffentlicht ist, in das Programm zu integrieren. Die Entwickler, die diese Komponente auswählten, befanden, dass diese Komponente die einzige Alternative für ihre Anforderungen ist. Dazu sei sie noch Open Source und kostenlos. Zwei Monate nach Veröffentlichung ihrer neuen Bildbearbeitungssoftware veröffentlichte Apple auch eine neue Entwicklerumgebung und eine neue Bildbearbeitungssoftware, die sich ein Patent zu Nutzen machte, das auf die MixIT GmbH registriert ist. Nach Konsultation eines Anwaltes musste die MixIT GmbH von einer Klage gegen Apple absehen, da sich Apple die starke Retaliation Klausel zu Nutzen machte. Die MixIT GmbH muss sich jetzt entweder nach einer alternativen Implementierung ohne Apple's

Open Source Lizenz umsehen oder man akzeptiert das entstandene Cross Licensing.

5.4.3 Beispielszenario 3: LibertatemVincIT GmbH

Die LibertatemVincIT GmbH integriert in ihrem Softwareprodukt nur Open Source Software und veröffentlicht ihr eigenes Produkt auch unter der GPL Lizenz. Für ihr Softwareprodukt benötigte LibertatemVincIT eine Komponente zur Speicherverwaltung, die sich einen ausgeklügelten Algorithmus zu Nutze macht. Diese Komponente steht unter der GPL Lizenz Version 1. LibertatemVincIT dachte, da in der Lizenz keine Rede von Patenten sei, seien diese auch lizenziert. Jedoch meldete sich wenig später eine andere deutsche Firma, die LibertatemVincIT der Patentverletzung anklagte und LibertatemVincIT aufforderte, die Nutzung des Patentes einzustellen. Zusätzlich zu dieser Anklage wurde diese Patentverletzung öffentlich, woraufhin das Softwareprodukt von LibertatemVincIT und damit die gesamte LibertatemVincIT GmbH geächtet wurde. Das Unternehmen steht nun vor der Frage was zu tun ist, um die Ächtung wieder zu beseitigen und welchen Weg man in Hinblick auf die Patentklage gehen solle. Andernfalls würde LibertatemVincIT so stark darunter leiden, dass es zur Geschäftsaufgabe kommen könnte.

5.5 Zusammenfassung

In diesem Kapitel wurden die Risiken, die mit der Verwendung von Open Source Software in einem eigenen Softwareentwicklungsprojekt in Verbindung stehen, betrachtet. Dazu wurden im Abschnitt 5.1 die juristischen Risiken für die Verwendung von Open Source Software erarbeitet. Besonders wurde dabei auf die Probleme mit fremdem Eigentum eingegangen und welche Folgen sich daraus ergeben können. Dazu wurden im Abschnitt 5.1.4 Beispiele aus der Praxis angegeben, die erläutern, welche Facetten derartige juristische Problemfälle annehmen können.
Im zweiten Abschnitt 5.2 wurden die finanziellen Risiken aufgezeigt und wie es schlussendlich bis zur Unternehmensaufgabe kommen kann. Die dort betrachteten Aspekte sind oft eine Folge von juristischen Problemen.
Im anschließenden Abschnitt 5.3 wurden die gesellschaftlichen Risiken und die damit verbundene Ächtung betrachtet. Es kann in der Softwarewelt zu großem Reputationsverlust kommen, der in Verbindung mit den finanziellen Risiken, die sich daraus ergeben,

zu einer Geschäftsaufgabe führen kann.
Der Absatz 5.4 führte Beispielszenarios der in Abschnitt 2.3 eingeführten Beispielunternehmen ein, die im nächsten Kapitel weitergeführt werden.
Es soll noch einmal darauf hingewiesen werden, dass die hier betrachteten Risiken sehr stark korrelieren und zusammenhängen. Eine genaue Trennung der einzelnen Aspekte lässt sich nicht finden, aber es lässt sich festhalten, dass die Auswirkungen in jedem Fall existenzbedrohend sein können.
Abschließend soll jedoch darauf hingewiesen werden, dass sich sowohl die GPL-Problematik, als auch die Patentproblematik meist ohne Schwierigkeiten umgehen lässt, wenn man nur findig und kreativ zu Werke geht [Ros04], [Man06]. Diese Problematik muss jedoch erst erkannt werden, dazu ist jedoch viel Open Source Kompentenz notwendig. Außerdem ist eine Umgehung der Gefahren auch nur durch zusätzlichen Ressourcenaufwand möglich.
Andere Wege, sich gegen derartige Risiken zu schützen, werden im nachfolgenden Kapitel 6 gezeigt.

6 Best practices in Unternehmen

Wie im vorangegangen Kapitel 5 geschildert, bestehen durch das Benutzen und Integrieren von Open Source Software in ein Softwareprodukt viele Risiken. Da es, wie bereits am Anfang der Arbeit beschrieben, bereits keinen Entwickler mehr gibt, der gänzlich auf den Einsatz von Open Source Software verzichtet, müssen diesen Risiken viele Vorteile gegenüber stehen. Einige seien hier genannt:

- Einfacherer Support mit vielen Updates [RE04b]
- Kosten und Verlässlichkeit [HH11]
- Flexibilität [HH11]
- Reduzierte Abhängigkeiten [HR]
- Innovation [All05]

Um eine möglichst große Effizienz in der Nutzung dieser Vorteile an den Tag zu legen, ist es die Aufgabe eines jeden Unternehmens, das auf Open Source Software setzt, die in Kapitel 5 aufgeführten Risiken zu mindern. Mit anderen Worten: Es ist nicht mehr die Frage, ob Open Source Software, sondern wie man dies umsetzt [Ham09]? Zu diesem Zweck gibt es verschiedene Methoden, die im weiteren Verlauf dieses Kapitels vorgestellt werden.
Im Abschnitt 6.1 wird zu diesem Zweck der Einsatz von Open Source in der Softwarelieferkette präzisiert, um ein besseres Verständnis von der konkreten Rolle von Open Source im Unternehmen zu erlangen. In Abschnitt 6.2 wird die Open Source Policy im Unternehmen angeführt. Dies gilt als Sammelbegriff für die in den darauf folgenden Abschnitten aufgeführten einzelnen best practices der Unternehmen.
Abschließend werden die Beispielszenarien fortgesetzt und eine Zusammenfassung des Kapitels gegeben.

6.1 Open Source Supply Chain

Der Einsatz von Open Source Software in einer Softwarelieferkette weist einige Unterschiede zu der herkömmlichen Softwarelieferkette auf. Die grundsätzlichen Ziele, die ein Unternehmen verfolgt, bleiben dabei jedoch gleich.
So gewinnt Open Source Software in der Lieferkette an Bedeutung, da sich proprietäre Softwarekomponenten, die man in sein Softwareprodukt integrieren möchte, schwer spezifizieren lassen. Open Source Software jedoch lässt sich durch den Zugang zum Source Code leichter spezifizieren [GS03]. Dadurch lässt sich genau die Software, die man benötigt, auswählen. Für diese Entscheidung spielen das Komponentendesign und die Architekturvereinbarkeit eine Rolle [GS03].
Ein weiteres Risiko, das in der proprietären Softwarelieferkette vorherrscht, besteht darin, dass der Softwarezulieferer, der eine Komponente des eigenen Softwareproduktes entwickelt hat, in die Insolvenz gerät. Dies würde in der proprietären Softwareentwicklung ein großes Risiko bedeuten, denn man müsste sich darauf verlassen, dass ein anderer die Entwicklung fortführt, oder man akzeptiert die Stilllegung der Benutzung beziehungsweise das Fehlen von weiteren Produktverbesserungen. Es kann auch in der Open Source Entwicklung zur Zahlungsunfähigkeit und Stilllegung von kommerziellen Open Source Entwicklern kommen. Durch den Zugang zum Source Code ist es durch intensives Einarbeiten in den Code aber auch möglich die Entwicklung mit eigenen Ressourcen im eigenen Haus fortzuführen. Was übrig bleibt, ist das NotInventedHere-Syndrom [Voa98].
Einen Unterschied zur proprietären Softwarelieferkette stellen auch die Wege, durch die Software in das Unternehmen findet, dar. So ist es durch das Allgegenwärtig-Sein von Open Source Software möglich, dass sie komplett unwissentlich für die Verantwortlichen in ein Softwareprodukt gelangt. Dies gilt im Speziellen für Community basierte Open Source Software, die meistens kostenlos aus dem Internet heruntergeladen werden kann [OFM+03]. Des Weiteren gibt es jedoch auch die üblichen Akquirierungsmethoden über Zulieferer und Verträge mit diesen. Dabei fällt die Beziehung zwischen Open Source Zulieferern oftmals sehr lose aus, was auch darin begründet ist, dass diese bei kostenloser Software nichts von der Integration mitbekommen [JBF07].
Ein weiterer Unterschied ist die Einschätzung der Total Cost of Ownership. Auf den ersten Blick ist der Kostenfaktor der Software durch die meistens sehr günstigen Anschaffungskosten sehr gering. Bei genauerer Betrachtung schlagen jedoch auch eventuelle Architekturwechsel und zusätzliche Ressourcen zu Buche, die die Total Cost of Ownership

von Open Source Software höher als erwartet ausfallen lassen [HPL05], [Gie12], [Eil08].

6.2 Open Source Policy

Eine Policy ist eine Menge von Regeln und Richtlinien [Bla12b]. Diese müssen leicht verständlich und einfach sein. Eine Open Source Policy ist demnach eine Sammlung an bestimmten Regelungen und Richtlinien, die sich mit Open Source befassen. Das Hauptaugenmerk wird dabei auf die Reduzierung von potentiellen Risiken gelegt, die durch das Verwenden von Open Source Software in einem Unternehmen entstehen können (siehe Abbildung 6.1).

Die Policy kann in jedem Unternehmen eine unterschiedliche reale Ausprägung haben,

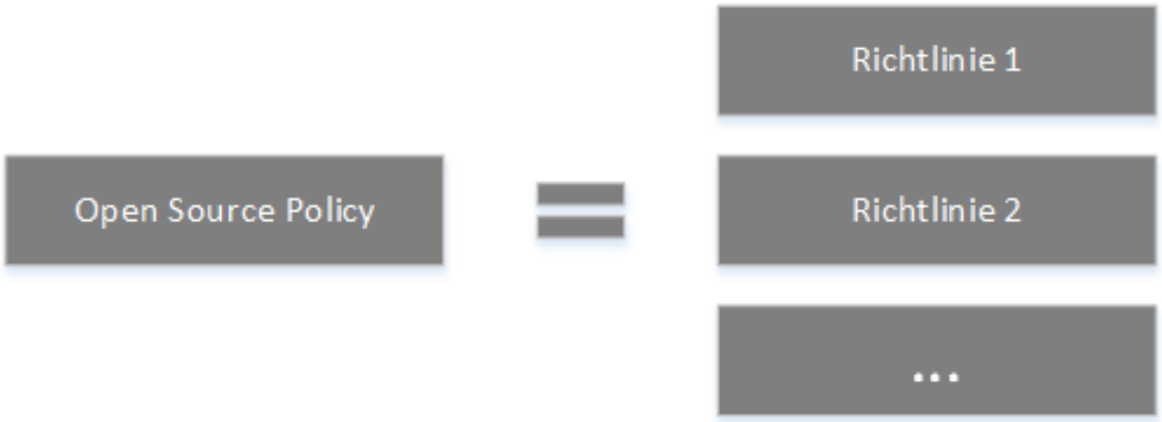


Bild 6.1: Veranschaulichung der Open Source Policy

da jedes Unternehmen den Umgang mit Open Source Software anders handhabt und handhaben will, beziehungsweise andere Anforderungen an die Nutzung von Open Source Software stellt. Die meisten, der hier vorgestellten Regelungen sind deshalb von abstrakter und wenig konkreter Art, da die Instantiierung dieser Maßnahmensschablonen unterschiedlich sein kann. Trotzdem gibt es einige Regelungen, die Unternehmen unbedingt umsetzen sollten, um sich gegen vorherrschende Risiken zu schützen.

Einige dieser Maßnahmen und Regelungen werden in den folgenden Abschnitten vorgestellt. Dazu werden zu Beginn jedes Abschnitts die Quellen für die Maßnahmen angegeben. Anschließend werde die in den Quellen vorgefundenen Angaben beschrieben und, wenn nötig, mit sinnvollen Ergänzungen versehen.

6.2.1 Code Label

Eine gute Vorgehensweise für das Managen von Open Source Code im Unternehmen ist das Einführen von einem Code Label [Bla12a]. Dieses Label zeigt an, welche Lizenz in einer Open Source Softwarekomponente verwendet wird und, wenn es mehrere Lizenzen sind, zu welchen Teilen welche Lizenz zum Einsatz kommt. Dabei werden, wie erwähnt, vornehmlich Softwarekomponenten dargestellt. Jedoch ist eine Darstellung auch auf unterschiedlichen Granularitätsstufen denkbar. So kann das Label auch für einzelne Softwaremodule oder komplexe Softwareprodukte erstellt werden. Daraus lassen sich viele Vorteile ableiten [Bla12a]:

- Informationen auf einen Blick einsehbar
- Eine einheitliche bildbezogene Sprache, die verschiedene Unternehmensteile verstehen
- Reduzierung von Geschäftsrisiken, weil Code Transparenz gefördert wird
- Beschleunigung der time-to-market durch einfachere Verhandlung mit Zulieferern oder Kunden

Dabei gibt es zwei denkbare Ansätze in Bezug auf die Verantwortlichkeit zum Vergeben des Code Labels. Das Code Label kann demnach von jedem Unternehmen selbst oder von einer unabhängigen Institution vergeben werden. Wenn das Unternehmen selbst die Verantwortung dafür besitzt, sollte eine objektive Methode bestehen, dass das Code Label vollständig, konsistent und der Wahrheit entsprechend ist. Denkbare Möglichkeiten dafür sind zum Beispiel standardisierte Tools oder eine umfangreiche Dokumentation. Eine unabhängige Institution gewährleistet durch einen objektiven Analyseprozess, der auf standardisierten Bewertungsgrundlagen basiert, Vergleichbarkeit und Objektivität. Desweiteren ist eine Erweiterung der Label um zusätzliche Eigenschaften der Softwarekomponenten denkbar. Dies könnten zum Beispiel Programmiersprache, Komplexität oder eingearbeitete Softwaremuster sein.

6.2.2 SPDX

Ein Projekt, das zur Einhaltung der Regeltreue mit Open Source Software von der Linux Foundation eingeführt wurde, ist das SPDX [Fou13]. SPDX liegt eine Spezifikation zu Grunde, die ein Standard Format definiert, mit dem Komponenten, Lizenzen

und urheberrechtlich geschützte Softwarepakete zwischen Unternehmen innerhalb einer Software-Zulieferkette ausgetauscht werden können.
Das Ziel der Spezifikation ist, dass die Einhaltung von Open Source Lizenzen forciert wird. Dazu wird versucht, die Informationsverteilung in der Lieferkette diesbezüglich einheitlich zu gestalten. Dadurch wird sowohl redundante Arbeit vermieden, als auch das Verständnis für den Gebrauch von Open Source Software innerhalb der Unternehmen und in der Lieferkette verbessert. Dem Kunden eines Zulieferers wird demnach eine Liste der verwendeten Komponenten und der einbezogenen Lizenzen geliefert.
Der Gebrauch der SPDX Spezifikation ist nicht verpflichtend. Es kann jedoch Unternehmen geben, die innerhalb ihrer Lieferkette darauf bestehen, dass ein SPDX Dokument ausgetauscht wird. Dies kann in den Verträgen in der Lieferkette verankert werden.
Die Spezifikation des SPDX Formats wird von einer unabhängigen Arbeitsgruppe der Linux Foundation erarbeitet. Dieser gehören viele Mitarbeiter von verschiedenen Open Source Firmen, Entwicklern und Kunden an. Der Prozess der Erarbeitung der Spezifikation ist weitgehend vergleichbar mit der Erstellung eines Open Source Softwareprodukts.
In der Praxis sieht die Verwendung von SPDX Dokumenten folgendermaßen aus:

- Jedem Softwarepaket liegt ein SPDX Dokument bei. Bei Update des Pakets muss auch das Dokument aktualisiert werden.
- Das SPDX Dokument umfasst Lizenzinformationen verknüpft mit jedem Dokument im Softwarepaket.
- Es kann sowohl für Open Source Software als auch für proprietäre Software erstellt werden.
- Es muss sichergestellt werden, dass das SPDX Dokument präzise ist (Historie der Autoren, Garantien in Verträgen, Automatisierung mit Werkzeugen).
- Es werden alle Lizenzen unterstützt, die von der Open Source Initiative genehmigt sind.

Die Intention des SPDX Formats ist demnach die gleiche, wie die oben aufgeführten Code Label. Der Schwerpunkt beider Methoden ist, dass die Verwendung von Open Source transparent und für jeden Stakeholder in der Software Lieferkette nachvollziehbar und bewertbar sein soll. Durch die Ähnlichkeit beider Ansätze werden diese im folgenden Kapitel als eine Methode zusammengefasst.

6.2.3 Interne Open Source Definition und Zielsetzung

Wichtig ist, dass das Unternehmen für sich eine Open Source Definition und eine Zielsetzung für Open Source Software aufsetzt [Ham09], [FAK04], [Duc13]. Dadurch kann von vornherein eine Bündelung der Ressourcen von statten gehen, um zum Beispiel Kosten zu sparen oder sich auf das eigene Entwickeln von Open Source Software zu konzentrieren. Wenn die Zielsetzung im Unternehmen nicht klar ist, werden Ressourcen, sowohl personeller als auch finanzieller Art falsch investiert. Dies kann durch Verschwendung von Arbeitszeiten der Entwickler passieren, indem sie sich auf die Beteiligung in Open Source Softwareentwicklung konzentrieren und das entstandene Produkt eine untergeordnete Rolle im Komponentenportfolio des Unternehmens spielt. Eine Definition von Open Source ist dahingehend wichtig, dass jeder Unternehmensmitarbeiter das eine und gleiche Verständnis entwickelt, was unter Open Source verstanden wird. Dadurch reduziert sich das Risiko, dass aus Unwissenheit Open Source Code in das Unternehmen gerät.
Die Aufgabe von Open Source Definitionen beziehungsweise Zielsetzungen in das Unternehmen zu bringen, obliegt dem Management. Dies kann jedoch in verschiedenen Unternehmen auch aus unterschiedlichen Ressorts der Unternehmensleitung geschehen. Es ist demnach nicht wichtig, ob die Open Source Vorgaben aus der Entwicklungsleitung oder aus der Informationstechnologie (IT)-Leitung stammen. Es ist nur darauf zu achten, dass sich das Management auf ein gemeinsames, konsistentes Vorgehen einigt und dieses auch vorlebt. Die Verteilung der Management Vorgaben kann mit Hilfe von Schulungen von statten gehen. Anschließend trägt jeder Mitarbeiter die Verantwortung die Ziele umzusetzen.
Einhergehend mit der Zielsetzung und der Definition von Open Source Software im Unternehmen sollte auch ein Open Source Prozess definiert werden, der Verantwortlichkeiten und Aktivitäten rund um Open Source definiert. Zum Beispiel sollte der Prozess beinhalten was geschieht, wenn Open Source Software von der Entwicklungsabteilung verwendet werden soll: Wer muss informiert werden? Wie sind die Voraussetzungen? Wie sehen die notwendigen weiteren Schritte aus[1]?

1 Es kann zum Beispiel von Belang sein, ob das Unternehmen ein Open Source Repository benutzt oder nicht. Wurde dieses vorher konsultiert?

6.2.4 Code Scanning und Code Audits

Eine effektive Methode sich vor Risiken durch Open Source Integration zu schützen, ist das Durchführen von Code Scans und Code Audits [Ope11], [Ell11], [HR], [Bla12b]. Sobald eine Softwarekomponente in das Unternehmen gelangt, ist es wichtig, diese Komponente hinsichtlich ihrer Bestandteile zu analysieren. Außerdem ist es wichtig auch jede Komponente, die nach außen geht, zu analysieren und zu scannen. Somit spielt die Praxis von Code Scanning und Audits sowohl für Zulieferer als auch für deren Kunden eine Rolle.
Die Zielsetzung der Scans und Audits sind vielfältig und können unterschiedliche Ausprägungen haben:

- Identifizierung von Lizenzen
- Aufdeckung von Patenten
- Bestimmung von abgeleiteten Arbeiten

Dadurch können frühzeitig rechtliche und ökonomische Probleme identifiziert werden und die Entscheidungen daraus auf einer fundierten Grundlage getroffen werden. So werden zum Beispiel beim Scanvorgang Lizenzunverträglichkeiten entdeckt, Patentverletzungen aufgedeckt und vermieden und bei Komponenten, die zur Distributionszwecken gescant werden, kann überprüft werden, ob man sich einem Vergehen schuldig gemacht hat. Die Folgen können sein:

- Nichteinbeziehung der Komponente unter den spezifizierten Lizenzbedingungen
- Bisher fehlende Patentlizenzen nach-erwerben
- Produzierten Code für die Distribution sperren
- Überarbeitung des eigenen Softwareproduktes
- Lizenzänderung des eigenen Softwareproduktes

Ein Problem mit den Code Scans besteht darin, dass diese bei Binärdaten nicht funktionieren, weshalb der Source Code verfügbar sein muss. Somit ist der Scan für die meisten proprietären Softwareprodukte nicht anwendbar.
Der Scan wird in den meisten Fällen durch ein spezielles Softwarewerkzeug durchgeführt. Dieses Werkzeug analysiert die Bestandteile der Softwarekomponente oder des Softwareproduktes und liefert anschließend einen Report über die vorgefundenen Merkmale. Als Referenz für das Tool sind viele Datenquellen mit Sammlungen von Patenten oder

bereits bestehenden Softwarelösungen verfügbar, die in Echtzeit vom Tool konsultiert und abgefragt werden können. Eine dieser Datenquellen ist zum Beispiel das bekannte Open Source Repository Sourceforge.net. Das automatisierte Scannen ist damit ein relativ einfach zu handhabender Prozess, da die eigentliche Aufgabe nur in der Zuführung von Source Code besteht. In Ausnahmefällen ist es jedoch auch notwendig eine manuelle Analyse der ein- beziehungsweise ausgehenden Source Codes durchzuführen. Dieser Fall ist jedoch sehr aufwändig und verlangt eine hohe Expertise des Analyseverantwortlichen. Wie oben beschrieben ist das Endprodukt eines Scanvorgangs ein Report, der Aufschluss über den Source Code gibt. Im besten Fall liegen keinerlei Verletzungen oder sonstige Besonderheiten vor. Falls jedoch Entscheidungen zu fällen sind, sind diese vom Open Source Verantwortlichen beziehungsweise von der im Unternehmen verankerten Open Source Organisation zu treffen. Wie bereits oben erwähnt, können die Folgen unterschiedlich sein.
Audits untersuchen anders als Scans nicht den ein- und ausgehenden Source Code, sondern begutachten die bereits im Unternehmen befindlichen Softwareitems. Dabei wird eine komplette Inventur der im Einsatz befindlichen Softwareprodukte erstellt und auf eventuelle Gefährdungspotentiale hinsichtlich Open Source Software untersucht. Damit können eventuelle Risiken, die mit der Benutzung oder Veränderung von Open Source Produkten einher gehen, von Vornherein vermieden oder abgeschwächt werden. Wichtig können Audits auch in Hinblick auf Unternehmensübernahmen sein, da für das Aufkaufen einer Firma eine lückenlose Buchhaltung über die eingesetzte Software den Wert des Unternehmens steigern kann.
Innerhalb eines Unternehmens werden diese Audits entweder von dem jeweiligen Softwareverantwortlichen durchgeführt oder die im Unternehmen aufgebaute Open Source Organisation übernimmt die Verantwortung. Es ist jedoch auch denkbar, dass spezialisierte Drittanbieter das Audit durchführen. Durch eine zentralisierte Aufbewahrung der Softwarelösungen kann ein effektives Audit unterstützt werden. Wie bei Scans werden bei Audits auch Softwarewerkzeuge verwendet, die automatisch das bestehende Softwareportfolio analysieren.

6.2.5 Training und Schulungen

Wichtig für den Erfolg aller Bemühungen für einen sicheren Umgang mit Open Source ist das Einführen von Schulungen und Trainings für Open Source Software im Unternehmen [HH11], [HP07], [Ell11], [FAK04]. Mit Hilfe dieser Schulungen lässt sich die im Unterneh-

men erarbeitete Definition für Open Source Software kommunizieren und verinnerlichen. Dort lässt sich auch in einem speziellen Rahmen und zugeschnitten auf das jeweilige Publikum das Risiko durch den Umgang mit Open Source Software darstellen. Es ist wichtig, dass diese Schulungen sowohl von Entwicklern als auch von Mitarbeitern der Rechtsabteilung, aber auch von Managern besucht werden, so dass sichergestellt ist, dass jeder Stakeholder umfassend über die Konsequenzen seines Handelns informiert ist.
Diese Schulungen können durch qualifizierte Coaches, die von Beratungsfirmen bestellt werden oder durch Experten innerhalb des Unternehmens durchgeführt werden. Sind solche Experten innerhalb des Unternehmen verfügbar, dann hat dies den Vorteil, dass Open Source Kompetenz dauerhaft im Unternehmen ist. Demgegenüber stehen die unabhängigen Coaches, die die Erfahrung von Open Source Einsatz aus verschiedenen Unternehmen mitbringen. Diese Erfahrung beinhaltet sowohl best practices als auch abschreckende Beispiele von Open Source Integration.
Inhalte der Schulungen sollten die Grundlagen von Open Source Software sein. Zusätzlich sollten die Grundlagen der Lizenzmechanik, Risiken der Lizenzen, Risiken durch Open Source, Grundlagen der Zusammenwirkung von Open Source und geistigem Eigentum, aber auch die Vorteile von Open Source und die richtige und risikoarme Nutzung von Open Source Software vermittelt werden. Wenn, wie oben erwähnt, alle Stakeholder Schulungen besuchen sollen, sollten bei jeder Stakeholder Gruppe unterschiedliche Schwerpunkte gesetzt werden, die dem späteren Umgang mit Open Source Software entsprechen.
Da das Umfeld von Open Software kontinuierlich weiterentwickelt wird[1], sollten einmal besuchte Schulungen wiederholt werden. Es ist auch darauf zu achten, dass die Inhalte der Schulung aktuellen Gegebenheiten und Entwicklungen entsprechend angepasst werden, um eine aktuelle und der Praxis entsprechende Vermittlung des Wissens zu ermöglichen.
Zusätzlich zu den eher theoretischen Schulungen sollten im Unternehmen auch hands-on Trainings angeboten werden. Innerhalb dieser Trainings werden Beispielszenarios durchgespielt. Diese Szenarios sind realen Situationen in Unternehmen nachempfunden und sollen damit einer leichteren Umsetzung der Theorie dienen, wobei noch keine realen Bedingungen und Risiken vorherrschen. Diese Trainings können durch Gruppenübungen oder Case Study Aufgaben umgesetzt werden.

1 Dies betrifft vor Allem das rechtliche Umfeld, da hier viele Änderungen von Gesetzen und Lizenzklauseln durchgeführt werden.

Das Ziel sollte sein, dass im Nachhinein jeder Stakeholder in der Lage ist, fundierte Entscheidungen zum Wohle des Unternehmens zu treffen und die damit verbundenen Risiken mit den Vorteilen abwägen zu können.

6.2.6 Einbeziehung der Rechtsabteilung

Die heutige Rechtssprechung in Bezug auf Software wird immer komplizierter (siehe Abschnitt 5.1.4), so dass es für Entwickler nicht mehr möglich ist, vollends über die rechtlichen Rahmenbedingungen informiert zu sein. Deshalb ist die Einbeziehung der Rechtsabteilung eine wichtige Maßnahme [Bla12b], [RE04b], [Bad10], [Ham09]. Diese Integration der Rechtsabteilung sollte sehr früh bei der Integration von Open Source Software geschehen, da unter Umständen schon sehr frühzeitig Risiken vermieden werden können. Dabei wird die Frage ob eine Komponente benutzt werden **kann** von Entwicklern getroffen und ob man es **darf** von der Rechtsabteilung. Es ist jedoch darauf zu achten, dass die Rechtsabteilung nicht mit jeder Kleinigkeit um Klärung beauftragt wird, da sonst der eigentliche Prozess unnötig verlangsamt wird. Dazu sollte durch die Rechtsabteilung eine Entscheidungsmatrix angefertigt werden. Auf deren Grundlage sollen Entwickler befähigt werden, eigene Entscheidungen schnell und sicher zu treffen, ohne zusätzlich Risiken einzugehen.
Jedoch sollte bei Unklarheiten eine Risikoklärung an die Rechtsabteilung erfolgen, da zum Beispiel durch die Retaliation Klausel ein großes Risiko für das Unternehmen besteht. Risiken können zum Beispiel der Verlust von geistigem Eigentum, der Offenlegung des Codes oder die Verletzung von Patenten oder Copyright sein. Um diesen Beratungsprozess der Rechtsabteilung in Hinblick auf Open Source Software umfassend zu ermöglichen, ist eine Schulung und Weiterbildung des Rechtspersonals unumgänglich. Außerdem ist es denkbar, dass ein spezieller Posten innerhalb der Rechtsabteilung speziell für Fragen rund um das Thema Software oder Open Source Software eingerichtet wird. Dadurch können Ressourcenengpässe vermieden werden. Außerdem wird den bestehenden großen Risiken, die mit (Open Source) Software einhergehen, Rechnung getragen.
Eine wichtige Entscheidung, die von der Rechtsabteilung zusammen mit den Verantwortlichen der Entwicklungsabteilung zu treffen ist, ist die Entscheidung ob der vorliegende und zur Diskussion stehende Code abgeleitete Arbeit ist oder nicht. Es ist jedoch selbst von Verantwortlichen der Rechtsabteilung schwer zu beurteilen, ob dies der Fall ist. Des Weiteren ist eine Unternehmensakquirierung, auch hinsichtlich des bestehenden Softwareportfolios der zu übernehmenden Firma durch die Rechtsabteilung abzuklären.

Zu diesem Zweck sollte ein Bericht über ein durchgeführtes Audit vorliegen. Das Audit sollte in diesem Fall sehr aktuell und vollständig sein, um keinerlei unkalkulierte Risiken bei Übernahme vorzufinden.

6.2.7 Open Source Software Repositories

Die Integration von Open Source Softwarekomponenten in ein Softwareentwicklungsprojekt passiert oftmals spontan oder unwissentlich. Erst im Nachhinein wird oft entdeckt, dass eine Komponente bereits an anderer Stelle in Benutzung ist und als risikoarm eingestuft ist. Deshalb ist eine Maßnahme, um neue Risiken durch Open Source Softwarekomponente zu vermindern, das Führen eines Open Source Software Repositories [FAK04], [Ham09].
Dieses Repository soll für die Entwickler, die nach einer Komponente zum Integrieren suchen, als erste Anlaufstelle dienen. Dadurch wird zusätzliches Risiko vom Unternehmen abgewandt, indem immer neue Open Source Komponenten für die gleichen Zwecke in das Unternehmen gelangen. Es ist wichtig, dass die Komponenten und Produkte, die im Repository platziert sind, einen strengen und zuvor definierten Analyseprozess durchlaufen haben und damit in Hinblick auf die möglichen Risiken als sicher einzustufen sind. Verantwortlich für die Verwaltung des Repositories und des Analyseprozesses sollten die internen Open Source Verantwortlichen sein. Falls keine interne Open Source Organisation aufgebaut wurde, obliegt die Verantwortung für das Repository dem IT-Verantwortlichen. Der Analyseprozess sollte des Weiteren definieren, ob jede Version eines Softwareproduktes abgelegt werden soll oder nur die neueste Version. Außerdem sollte festgelegt werden, ob jede Version neu analysiert werden muss oder nicht. Dabei gibt es mehrere Möglichkeiten:

- Jede Version analysiert und jede Version einzeln abgelegt: Hat den Vorteil, dass kein Risiko eingegangen wird, aber hoher Analyseaufwand und Suchaufwand für den Entwickler entsteht.
- Nur Hauptversionen analysiert und jede Version einzeln abgelegt: größeres Risiko weil neueste Version keine Hauptversion sein muss, aber geringerer Analyseaufwand.
- Jede Version analysiert und nur die Aktuellste abgelegt: Das ist die sicherste Möglichkeit, aber hat hohen Analyseaufwand dafür jedoch geringen Suchaufwand für den Entwickler.

Zusätzlich muss die Granularität der Inhalte des Repositories festgelegt werden. Es ist denkbar, dass sowohl Softwaremodule, Softwarekomponenten als auch Softwareprodukte in das Repository gelegt werden können. Außerdem können dort auch Bibliotheken verwaltet werden. In Folge der Ablegung aller Open Source Softwareteile sollten auch die Zusammenhänge dieser Teile dokumentiert werden. Dies geschieht mit der Intention Lizenzen besser zu entsprechen.
Um die Suche innerhalb dieses Speichers zu erleichtern, sollte eine einfache und anschauliche Darstellung der Komponente inklusive der Art der Lizenz, wichtiger technischer Details und wichtiger entscheidungsfördernder Hinweise aufgeführt werden. Standardmäßig sollte eine grafische Oberfläche beziehungsweise webbasierte Oberfläche angeboten werden.

6.2.8 Akquirierungs- und Trackingprozess für Open Source Komponenten

Eine weitere Maßnahme gegen die Risiken ist das Definieren eines Open Source Einkaufsprozesses [Ell11], [HP07], [Ham09]. Es ist wichtig, dass dieser Prozess für alle Arten von Software herangezogen wird. Dieser Prozess soll die finanziellen Aspekte eines Softwarekaufs in vollem Umfang bewerten. Zusätzlich werden die am Markt oder im Unternehmen vorhandenen Alternativen berücksichtigt. Des Weiteren sollte der Prozess beinhalten, wie wichtig die Komponente für das spätere Endprodukt ist. Handelt es sich um einen integraler Bestandteil oder ist es ein Bestandteil, der leicht ersetzbar ist. Dazu sollte eine Bewertung der zur Verfügung stehenden Infrastruktur, Ressourcen und sonstigen Rahmenbedingungen einbezogen werden. Die Bewertung und die Definition des Prozesses sollten durch die internen Open Source Verantwortlichen in Verbindung mit dem Einkaufsverantwortlichen geschehen, um beide Standpunkte angemessen und konsistent zu berücksichtigen.
Damit lässt sich anschließend eine fundierte und risikomindernde Entscheidung für das Unternehmen treffen. Zusätzlich sollte, bei erfolgreicher Beendigung des Kaufprozesses, ein Trackingprozess für Open Source Software angestoßen werden. Dieser verfolgt die Benutzung der Komponente durch den kompletten Lebenszyklus der Software. Dadurch ist es möglich, abgeleitete Arbeit oder sonstige lizenzrechtlich problematischen Dinge, sofort signalisiert zu bekommen. Wichtig für diesen Vorgang ist das Dokumentieren. Jede Änderung an einer getrackten Komponente muss dokumentiert werden, um eine

Änderung im Nachhinein noch nachvollziehen zu können.
Da durch die Vielzahl im Unternehmen vorhandener Softwarekomponenten ein manueller Trackingprozess unmöglich ist, sollte für eine umfassende Werkzeugunterstützung und der damit einhergehenden Automatisierung gesorgt werden. Eine wesentliche Grundlage für die Automatisierung und das Tracken ist, dass jede Softwarekomponente durch den Lebenszyklus eindeutig identifizierbar ist. Deshalb ist jeder Komponente eine eineindeutige Identifikationsstruktur zu Grunde zu legen.
Die Verantwortlichkeit für das Überwachen des Trackingprozesses liegt beim Open Source Verantwortlichen, der bei Verletzung von Open Source Richtlinien die Folgen abschätzen muss und weitere Schritte einzuleiten hat.

6.2.9 Interne Open Source Verantwortlichkeiten

In vielen Unternehmen fehlt eine klare Verteilung der Verantwortlichkeiten, wer zuständig für das Bewerten und Analysieren der Risiken ist, die von Open Source Komponenten ausgehen. Deshalb ist es überaus wichtig, dass in den Unternehmen weitreichende organisatorische Änderungen vorgenommen werden [Ham09], [HP07], [Ope11], [FAK04], [Bla12b].
Dazu empfiehlt sich die Einrichtung von:

- Open Source Review Boards
- Open Source Compliance Verantwortlicher
- Open Source Program Office

Es ist Aufgabe der Unternehmensleitung diese Organisationsstrukturen einzuführen und zu kommunizieren. Dabei kann sich die personelle Zuordnung zu den Aufgaben überschneiden, sodass mehrere Kompetenzen auf eine Person verteilt werden.
Das Open Source Review Board hat die Aufgabe Entscheidungen zu treffen, wenn eine rechtliche Situation in Bezug auf Open Source Software nicht eindeutig ist. In diesem Review Board sollten Stakeholder aus verschiedenen Unternehmenszweigen sein, aber auch verantwortliche Manager, die die Kompetenz haben, weitreichende Entscheidungen zu fällen. Außerdem ist das Review Board verantwortlich für die Überwachung der Open Source Nutzung im Unternehmen. Falls es zu Verletzungen der vorgegeben Richtlinien kommt, ist das Review Board angehalten einzugreifen und eine Entscheidung zu fällen. Diese Richtlinien werden vom Open Source Program Office erarbeitet. Dort werden die

Richtlinien und Regelungen innerhalb des Unternehmens in Bezug auf Open Source getroffen. Daher ist es wichtig, dass es Entscheidungsträger innerhalb des Open Source Program sind, um eine reibungslose Umsetzung im Unternehmen durchzusetzen. Es ist die Aufgabe des Program Offices diese Richtlinien zu kommunizieren und zu veröffentlichen. Dies kann geschehen durch Schulungen, Trainings oder andere Kommunikationsplattformen, die im Unternehmen etabliert sind.
Der Compliance Verantwortliche ist zuständig für das Einhalten dieser Richtlinien und ist der erste Ansprechpartner für die Entwickler, die Fragen haben. Er ist derjenige, der dafür sorgt, dass Risiken auf unterer Ebene weitestgehend vermieden werden. Dieser Verantwortliche muss nicht nur eine einzelne Person sein. Er sollte in den oben genannten Gremien vertreten sein, um einen umfassenden Fokus auf das Open Source Programm des Unternehmens zu haben.
Durch das Einführen dieser Open Source Organisation in allen Unternehmensebenen wird auf allen Hierarchiestufen das Risiko gemindert. Zusätzlich wird das Wissen und das Risikobewusstsein auf jedem Level verteilt.
Falls die Expertise der internen Verantwortlichen für Open Source an ihre Grenzen stößt, ist es unumgänglich einen externen Berater zu konsultieren. Es sollte daher frühzeitig, bei Abzeichnung von Kompetenzlücken Kontakt zu externen Beratern hergestellt werden, um eine spätere, aufwändigere Einarbeitung in die Unternehmensstrukturen vorzuziehen.

6.2.10 Zuliefererbeziehung

Die Lieferantenbeziehung spielt ebenfalls eine Rolle zur Verminderung der Risiken durch Open Source Software [Ell11], [HH11], [Gro13]. Durch Transparenz und Zusicherung von Garantien kann eine risikoarme Integration von Open Source Komponenten sichergestellt werden. Eine Rolle in dieser Hinsicht spielt auch die Historie und allgemeine Reputation des Zulieferers. Wenn dieser bereits in der Vergangenheit durch Gerichtsfälle oder Ähnliches aufgefallen ist, wird dies die Wahrscheinlichkeit mindern, diesen Zulieferer in Betracht zu ziehen, aber gleichzeitig sinkt auch die Wahrscheinlichkeit dafür, dass eine Softwarekomponente in das Unternehmen gelangt. Und dies wiederum sorgt dafür, dass das Risiko, das von dieser Komponente ausgegangen wäre, nicht mehr relevant ist.
Diesen Entscheidungen sollte eine Entscheidungsmatrix zu Grunde liegen. Diese Matrix sollte von Open Source Verantwortlichen oder den Wettbewerbsanalysten des Unternehmens erstellt werden. In dieser Matrix wird festgehalten, welche Zulieferer positiv und welche negativ bewertet werden. Über die Zeit werden sich dadurch feste

Partnerschaften entwickeln, die nicht nur zur Risikoverminderung, sondern vor allem auch zur Verbesserung des Geschäfts beitragen werden.
Falls es unerlässlich erscheint eine Komponente von einem als schlecht bewerteten Zulieferer zu beziehen, muss eine Einschätzung und Abwägung durch den Open Source Verantwortlichen des Unternehmens erfolgen. Diese Einschätzung sollte auch beinhalten, wie eine risikoarme Verwendung der Komponente oder des Produktes sichergestellt werden kann oder ob von einer Verwendung gänzlich abzusehen ist.
Wichtig in Hinblick auf die Zulieferbeziehung ist auch, dass in Verträgen mit Zulieferern und Kunden wichtige vertragliche Details fixiert werden und diese im Falle einer Nichteinhaltung auch unter Sanktionen gestellt werden. Die Sanktionen können vielfältig sein und von Beendigung der Geschäftsbeziehung bis zu fest fixierten Geldstrafen reichen. Vertragliche Details können zum Beispiel die Zusicherung des verantwortungsvollen Umgangs mit Open Source Software sein oder die Garantie nur noch lizenzierte Patente zu benutzen.

6.2.11 Weitere Maßnahmen

In diesem Abschnitt werden noch weitere Maßnahmen vorgestellt. Diese Maßnahmen nehmen nicht unbedingt direkten Einfluss auf die Verhinderung von Risiken mit dem Umgang von Open Source Software in den Zulieferketten, können jedoch für das Unternehmen eine Hilfe sein die Folgen der Gefahren abzumildern. Außerdem sind die vorgestellten Maßnahmen nicht für alle Unternehmen anwendbar; für betroffene Unternehmen jedoch ein wichtiger Schutz.

6.2.11.1 Open Source Versicherung

Eine Möglichkeit sich vor den Folgen der falschen Open Source Software Verwendung zu schützen, ist das Schützen durch Versicherungen [Gro13]. Durch diese Versicherungen lässt sich nicht nur gegen normale Geschäftstransaktionen mit Open Source Software absichern, sondern auch gegen die Gefahren, die durch Geschäftsübernahmen entstehen. Diese Versicherungen können folgende Gefahren abdecken:

- Gewinneinbußen durch Gerichtsverfahren, die ein Weiterverkaufen von Produkten unmöglich machen
- Ausgleichen von verminderten Kaufpreis durch falsche Geschäftsübernahme Modularitäten

- Kosten zum Reparieren und Ersetzen von Code, um den Anforderungen zu genügen

Durch das Abschließen einer Versicherung lässt sich das Risiko einer Open Source Verletzung nicht minimieren, sondern nur die Folgen abschwächen. Deshalb muss es das Ziel jedes Unternehmens sein, durch Implementierung der oben vorgestellten Maßnahmen eine Versicherung obsolet zu machen.

6.2.11.2 Kommunikation mit Community

Für Unternehmen, die über eine an der Entwicklung des Softwareproduktes beteiligten Community verfügen, ist die Kommunikation mit dieser Entwicklergemeinde wichtig [Duc13]. Diese Maßnahme ist nur für Unternehmen relevant und praktikabel, die über eine Entwicklergemeinde verfügen, die aktiv im Open Source Prozess involviert ist. Von dieser Community können dementsprechend die gleichen Gefahren ausgehen wie von Entwicklern innerhalb des Unternehmen, nur diese Entwickler sind nicht so einfach zu kontrollieren, weil sie auf der ganzen Welt und unter Umständen anonym verteilt sind.
Es ist daher unabdingbar für solche Unternehmen eine zielgerichtete Kommunikation mit der Community zu etablieren. Dabei sollten die selben Kommunikationsziele hinsichtlich der Vermeidung von Risiken durch Open Source ausgegeben werden wie auch innerhalb des Unternehmens, um die Intention des Unternehmens auch in die Köpfe der verteilten Entwickler zu bringen.
Es ist für die Unternehmen sehr wichtig, dass das Code Scanning für den eingehenden Code aus der Community einwandfrei funktioniert, um sich auf diesem Wege gegen etwaige böswillige Attacken, Unwissenheit der Entwickler oder Nachlässigkeiten zu wappnen.
Desweiteren können auch Trainings und Schulungen für Communitymitglieder angeboten werden, die die Open Source Strategie des Unternehmens darstellen und auf die Risiken hinweisen. Gedacht sind die Trainings und Schulungen vornehmlich für besonders aktive Communitymitglieder.

6.2.11.3 Sicherheit von Open Source Code

Wichtig für die weitere Verwendung von Open Source Software ist das Sicherstellen der Sicherheit der Software [HP07]. Es ist wichtig, dass durch Sicherheitsvorkehrungen wie Firewalls oder Code Scannern das Einbringen von Malware, Spyware und sonstigen

Schadprogrammen in das Softwareprodukt verhindert wird.
Die Distribution von schadhaften Produkten kann die Reputation des Unternehmens schädigen, was in der Folge auch zu großen finanziellen Einbußen führen kann. Dabei entsteht die Gefahr nicht unbedingt durch den Einsatz und die Verwendung von Open Source Software, aber die Gefahr ist in diesem Umfeld potentiell höher, da der Code für jeden einsehbar ist.

6.3 Veranschaulichung am Fallbeispiel

In diesem Teilstück wird wieder auf die oben genannten Szenarien der drei Beispielfirmen eingegangen, um zu veranschaulichen welche Möglichkeiten die Unternehmen haben, um sich vor den jeweiligen Gefahren und Risiken zu schützen.

6.3.1 Beispielszenario 1: PropForSuccess GmbH

PropForSuccess hat sich aus einem Pool von proprietären Softwarekomponenten bedient und dabei einen Code mit GPL lizenziertem Code entdeckt. Daraufhin hat sich PropForSuccess entschieden eine alternative Software zu suchen, da eine Open Source Strategie nicht akzeptabel für das Management war. Man hat sich des Weiteren dazu entschlossen, die Suche nach einer neuen Komponente dahingehend zu erweitern, dass die Reputation des Zulieferers der Komponente eine wichtige Rolle spielt. Die GPL lizenzierte Komponente kam von einem Zulieferer, der schon des Öfteren durch rechtliche Zwischenfälle negative Presse bekommen hat, sein Produkt jedoch preislich weit unter Marktniveau angeboten hat. Aus diesem Grund hat man sich im ersten Anlauf auch für diesen Zulieferer und dessen Produkt entschieden. Damit wird in Zukunft durch das sensiblere Auswählen von Zulieferern versucht frühzeitig Gefahr signalisiert zu bekommen und das erneute Risiko einer Lizenzverletzung zu vermeiden.

6.3.2 Beispielszenario 2: MixIT GmbH

MixIT ist in einer schwierigen Lage. Die Komponente, die sie eingebaut haben, weist das spezifizierte und geforderte Verhalten auf, das für den Bestandteil ihres eigenen Softwareproduktes von Nöten ist, jedoch benutzt Apple ein eigenes Patent, um damit im

Endeffekt Marktanteile von MixIT zu übernehmen.
Aber wie hätte man die Situation verhindern können? Grundsätzlich hätte MixIT nicht einfach eine Open Source Komponente unter Apple Lizenz integrieren können. Das Problem bestand aber darin, dass der zuständige Entwickler nicht wusste, dass in der Apple Lizenz eine Retaliation Klausel vorhanden ist, und selbst wenn er es gewusst hätte, hätte er auch wissen müssen, was dies für das eigene geistige Eigentum bedeutet. Um die Gefahr dieser Unwissenheit zu bekämpfen, bieten sich in erster Instanz Schulungen für die Entwicklung an, da dieser spezielle Entwickler dazu befähigt worden wäre eine andere Komponente unter anderen Bedingungen auszuwählen. Weiterhin ist es sinnvoll als zweite Instanz die Rechtsabteilung bei solchen komplizierten und komplexen rechtlichen Belangen einzubeziehen. Nach Konsultation des Anwaltes wäre das Offenlegen des geistigen Eigentums und der damit verbundenen Gefahr aufgedeckt worden.

6.3.3 Beispielszenario 3: LibertatemVincIT GmbH

LibertatemVincIT steht durch eine Patentverletzung in der Öffentlichkeit und sie wird deshalb stark kritisiert. Dass dies aus Unwissenheit und nicht aus Boshaftigkeit seitens LibertatemVincIT passiert ist, interessiert in diesem Fall keinen. Dass damit die Insolvenz für LibertatemVincIT verbunden sein kann, interessiert insoweit auch wenige, da das Unternehmen in ihren Augen sowieso keine Berechtigung hat auf dem deutschen Markt Produkte an den Mann zu bringen.
Dabei wäre eine Vermeidung dieser Problematik einfach gewesen. Durch einen Scan durch diese Komponente wäre entdeckt worden, dass ein Patent für diesen Algorithmus vorliegt und man sich darum kümmern muss die Lizenz dafür zu erlangen. Des Weiteren hätte eine Schulung im Unternehmen dafür gesorgt, dass bei Verwendung der GPL Lizenz Version 1 darauf zu achten sei, dass keine Patentlizenzen erteilt werden. Wenn man sich anschließend als Entwickler nicht mehr sicher fühlt, ist es wichtig, dass die Rechtsabteilung eingeschaltet wird.
Bei der LibertatemVincIT GmbH ging man den gleichen Weg wie man es aus dem Industrieunternehmen Siemens mit den Korruptionsskandalen kennengelernt hat: Man hat durch aktive Transparenz und Offenheit Besserungen auf den Weg gebracht und die Schuld eingestanden. Eine der Maßnahmen war es auch eine effektive Open Source Policy mit einer strukturierten Open Source orientierten Organisation aufzubauen.

6.4 Zusammenfassung

Dieses Kapitel gab eine Antwort auf die Risiken, die in Kapitel 5 vorgestellt wurden. Als erstes wurde die spezielle Situation von Open Source in der Lieferkette betrachtet, dabei wurden Unterschiede in der Lieferkette mit Open Source Software und der ohne Open Source Software betrachtet und dargestellt. Der nächste Abschnitt führte den Begriff der Open Source Policy ein. Dies ist ein Sammelbegriff, der für ein Unternehmen wichtige Regelungen für den Umgang mit Open Source beinhaltet. Im Anschluss wurden viele Maßnahmen zur Verhinderung oder Minimierung von Risiken vorgestellt. Dazu zählt die Vorstellung des Code Labels, was ermöglicht einen ersten wichtigen Überblick über die enthaltenen Open Source Komponenten zu erlangen. Weiterhin wurde beschrieben, dass es für ein Unternehmen wichtig ist eine eigene Vision und Mission im Umgang mit Open Source Software zu erlangen. Daraufhin wurden das Code Scanning und Audits vorgestellt, um eingehende als auch ausgehende Softwarebestandteile zu analysieren, um etwaige Risiken frühzeitig zu erkennen. Training und Schulung als nächste vorgestellte Maßnahme ist dahingehend wichtig, dass alle Stakeholder die mit Open Source Software Kontakt haben, auch wissen was sie tun und damit aktiv gegen Risiken kämpfen. Die nächste Maßnahme ist, dass die Rechtsabteilung bei strittigen beziehungsweise unklaren Situationen zu Rate gezogen werden soll, da sie über die nötige Expertise für die rechtlichen Rahmenbedingungen verfügt. Des Weiteren ist es wichtig einen neuen Kaufprozess für Open Source Software aufzusetzen, um damit schon beim Kauf der Software die Gefahren zu bekämpfen. In die gleiche Richtung geht auch die Maßnahme, die besagt, dass man bei der Wahl seiner Lieferanten bereits potentielle schwarze Schafe ausmacht und nicht für seine eigenen Komponenten verwenden sollte, um eigene Schäden zu vermeiden. Eine wichtige Maßnahme ist das Verändern und Ausrichten der Organisation der Unternehmung auf die speziellen Anforderungen von Open Source Software und der damit verbundenen Einrichtung von zusätzlichen Kompetenzträgern. Zum Abschluss wurden die in Kapitel 5.4 vorgestellten Beispielszenarien dahingehend erweitert, dass für die vorgestellten Risiken mindestens eine Methode der Vermeidung dieser Risiken vorgestellt wurde.

7 Vorschlag eines Open Source Zertifizierungsmodells in der Lieferkette

In diesem Kapitel wird ein Vorschlag für ein Open Source Zertifizierungsmodell in der Lieferkette vorgestellt. Dazu wird in Abschnitt 7.1 eine Motivation angegeben, die eine Antwort auf die Frage *Wozu benötigt man ein Zertifizierungsmodell in der Open Source Software Lieferkette?* gibt.
Anschließend werden in Abschnitt 7.2 die nötigen Konzepte des Modells eingeführt und erläutert. Danach wird der Prozess beschrieben, mit Hilfe dessen die Zertifizierung eines Unternehmens erfolgt (Abschnitt 7.4). Die Klassifizierung in unterschiedliche Stufen wird im Teil 7.3 angegeben. Abschließend wird die Integration in die Software Lieferkette veranschaulicht und insbesondere die Auswirkungen des Zertifizierungsmodells auf diese Lieferkette.

7.1 Motivation

In Kapitel 5 wurden sehr viele Risiken für das Einbinden von Open Source Software in die eigene Softwareentwicklung besprochen. Diese Risiken können viele verschiedene Ausprägungen haben. Das Äußerste was einem Entwickler beziehungsweise Unternehmen passieren kann, ist die Geschäftsaufgabe. Mindestens jedoch ist das Eingehen solcher Risiken mit gesellschaftlichen, ökonomischen und juristischen Folgen verbunden. Das Abschätzen dieser Gefahren gestaltet sich schwierig, da man sich in den meisten Fällen auf die Angaben des Entwicklers verlässt. Eine wichtige Maxime in dieser Hinsicht ist also das Vertrauen.
Um diese Gefahren zu umgehen, etablierten viele Unternehmen bestimmte Maßnahmen, die sich oft unter dem Begriff Open Source Policy zusammenfassen lassen. Dies wurde in

Kapitel 6 vorgestellt. Der Grund für diese Maßnahmen ist klar, denn kein Unternehmen will das große Risiko eingehen, das mit der Benutzung von Open Source in der Lieferkette einhergeht, ohne sein Möglichstes getan zu haben das Gefahrenpotential zu vermindern. Der Grundsatz lautet also: Dort, wo vorher erhöhtes Vertrauen unter den Zulieferern herrschte, regiert nun Kontrolle der Zulieferer durch Maßnahmenpakete der Unternehmen. Dies getreu dem Motto: Vertrauen ist gut, Kontrolle ist besser.
Die Schwierigkeit der oben genannten Open Source Policy besteht jedoch weiterhin darin, dass das damit benutzte Maßnahmenpaket unvollständig und von Unternehmen zu Unternehmen unterschiedliche Ausprägungen haben kann. Es kann daher passieren, dass man sich in Sicherheit fühlt, durch das fehlende Implementieren bestimmter Maßnahmen aber weiterhin ein hohes Gefahrenpotential besteht. Dieser Effekt wird umso größer je länger und komplexer die Lieferkette eines Unternehmens ist. Das Gefahrenpotential wird von einem Zulieferer der Stufe n zu einem Zulieferer n-1 weitergereicht, wenn beide keine geeignete und umfassende Open Source Policy betreiben.
Was kann also dagegen getan werden, diese Effekte zu vermindern? Welches Maßnahmenpaket bietet einen umfassenden Schutz? Können Standards in der Lieferkette etabliert werden, die sich die Verminderung des Gefahrenpotentials zum Ziel setzen? Wie kann Transparenz hinsichtlich des Risikopotentials eines Unternehmens erreicht werden?
Eine Antwort auf diese Fragen soll das nun vorgestellte Zertifizierungsmodell geben. Die groben Ziele des Modells lauten daher:

- Transparenz
- Einfachheit und Verständlichkeit
- Erhöhtes Vertrauen
- Risikoreduzierung hinsichtlich der Verwendung von Open Source

7.2 Konzept

Dieser Abschnitt stellt die verschiedenen Konzepte des Modells vor. Dazu werden zuerst allgemeine Rahmenbedingungen vorgestellt, anschließend die drei Zertifizierungslabel und der Maßnahmenkatalog eingeführt.

7.2.1 Allgemein

Wie oben bereits beschrieben, ist das große Ziel des Modells das Verwenden von Open Source in der Software Lieferkette kontrollierbar zu machen und damit das Risikopotential zu verringern. Weiterhin muss es das Ziel sein, dass die komplette Lieferkette einen möglichst großen Schutz gegen die Gefahren etabliert. Zu diesem Zweck wird das Zertifizierungsmodell vorgestellt. Dieses wird jeweils auf ein Unternehmen angewandt, also nicht auf die gesamte Lieferkette. Jedes Unternehmen ist selbst dafür verantwortlich das Modell zu implementieren.

7.2.2 Drei Zertifizierungslabel

Grundsätzlich gibt es im Zertifizierungsmodell drei verschiedene Arten von Label zu erlangen. Diese sind abhängig vom Grad der Nutzung von Open Source Software (siehe auch Abbildung 7.1).

Label 1: Das Unternehmen benutzt keinerlei Open Source Software

Label 2: Das Unternehmen integriert Open Source Software

Label 3: Das Unternehmen entwickelt selbst Open Source Software

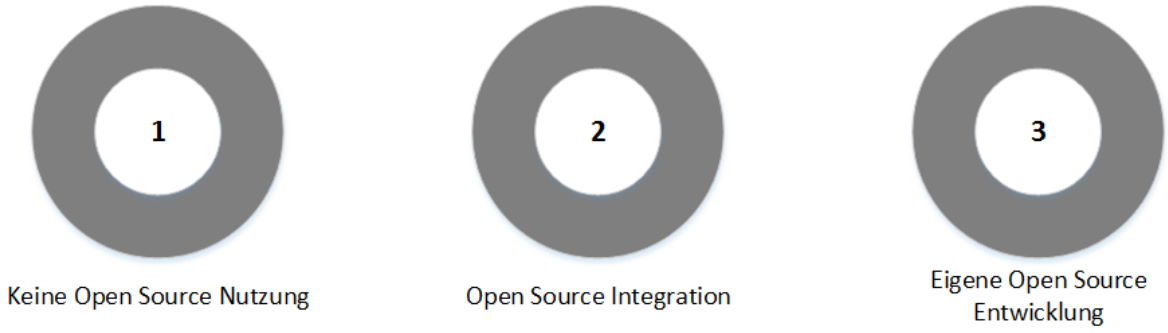


Bild 7.1: Die drei Label

7.2.2.1 Label 1

Auf den ersten Blick ist es verwunderlich, warum ein Unternehmen, das keinerlei Open Source Komponenten benutzt, eine Zertifizierung für Open Source in der Lieferkette durchführen sollte. Auf den zweiten Blick wird jedoch klar, dass selbst ein solches

Unternehmen den Risiken durch Open Source ausgesetzt ist. Denn auch wenn das Unternehmen glaubt, dass eine Komponente keine Open Source Komponente ist und dies obendrein noch vom Zulieferer bestätigt bekommt, kann es sein, dass dennoch ein Bestandteil Open Source Software in der Komponente enthalten ist. Dies kann zum Beispiel geschehen, indem der Zulieferer selbst von verschiedenen Zulieferern Software bezieht und diese Open Source Komponenten benutzten und dies nicht kennzeichneten. Das Ergebnis ist das potentielle Risiko, das bei jedem einzelnen Unternehmen besteht. Das Risiko ist sogar stellenweise ungleich höher, denn viele Open Source Lizenzen verbieten die kommerzielle Nutzung der lizenzierten Software. Die Folge wäre schlimmstenfalls das Offenlegen des Codes, mindestens jedoch die Verschwendung von Ressourcen, um die Lizenzverletzung zu umgehen.
Diese Unternehmen streben das Zertifizierungslabel 1 an.

7.2.2.2 Label 2

Die Unternehmen, die dieses Label anstreben, sind die Unternehmen, die keine eigene Open Source Software schreiben, jedoch Open Source Software verwenden. Es ist klar ersichtlich, dass sie den Risiken aus Kapitel 5 ausgesetzt sind, wenn keine geeigneten Maßnahmen ergriffen werden. Dass sie selbst keine Open Source Software entwickeln, kann bedeuten, dass sie entweder proprietäre Entwickler sind, die Open Source Komponenten benutzen oder sie sind kommerzielle Open Source Entwickler, die viele verschiedene Open Source Komponenten integrieren. Die Auswirkungen der Gefahren sind in jedem Falle schwerwiegend für die Entwickler.

7.2.2.3 Label 3

Dieses Label wird von Open Source Unternehmen angestrebt, die sowohl Open Source Komponenten in ihre Entwicklung integrieren als auch selbst Open Source Code schreiben. Auch hier ist es eindeutig, dass die Risiken aus Kapitel 5 für diese Unternehmen gelten.

Es ist wichtig zu beachten, dass eine Softwarelieferkette Unternehmen mit allen Labeln enthalten kann und diese Label keine Qualitätsabstufungen geben, sondern nur als Information und Hinweis auf die Verwendung von Open Source Software dienen.

7.2.3 Maßnahmenkatalog

Da die Einteilung in Labels in Abschnitt 7.2.2 noch kein Ranking hinsichtlich der Qualität der Kontrolle von Open Source liefert, wird hier der Maßnahmenkatalog eingeführt. Dieser Maßnahmenkatalog umfasst eine Auswahl von Maßnahmen, die zum Schutz vor Risiken dienen. In Kapitel 6 wurden verschiedene solcher Maßnahmen vorgestellt.
Die dort aufgeführten Maßnahmen sind jedoch in keinster Weise vollständig und erheben auch nicht den Anspruch redundanzfrei zu sein. Deshalb ist es wichtig, dass das Zertifizierungsmodell einen erweiterten und standardisierten Maßnahmenkatalog ermöglicht (siehe dazu Abbildung 7.2). Generell wird die Anzahl der Vorkehrungen im Maßnahmenkatalog über die Zeit hinweg steigend sein, denn die Komplexität und Anzahl der Risiken wird immer größer. Vor jeder Aufnahme in den Maßnahmenkatalog sollte auf Neuheit, Wirksamkeit und Redundanz geachtet werden, da der Maßnahmenkatalog ansonsten zu groß, zu unübersichtlich und irrelevant wird.
Der Maßnahmenkatalog dient als Grundlage der Bewertung der Kontrolle der Open Source Software Risiken im Unternehmen. Vor der Zertifizierung sollte eine einheitliche Fassung des Maßnahmenkatalogs sichergestellt werden und diese standardisierte Fassung sollte anschließend Anwendung in allen Unternehmen der Lieferkette finden.
Für die weitere Betrachtung in diesem Kapitel sollen die neun bereits vorgestellten Maßnahmen als Maßnahmenkatalog dienen:

1 Code Label und SPDX

2 Interne Open Source Definition und Zielsetzung

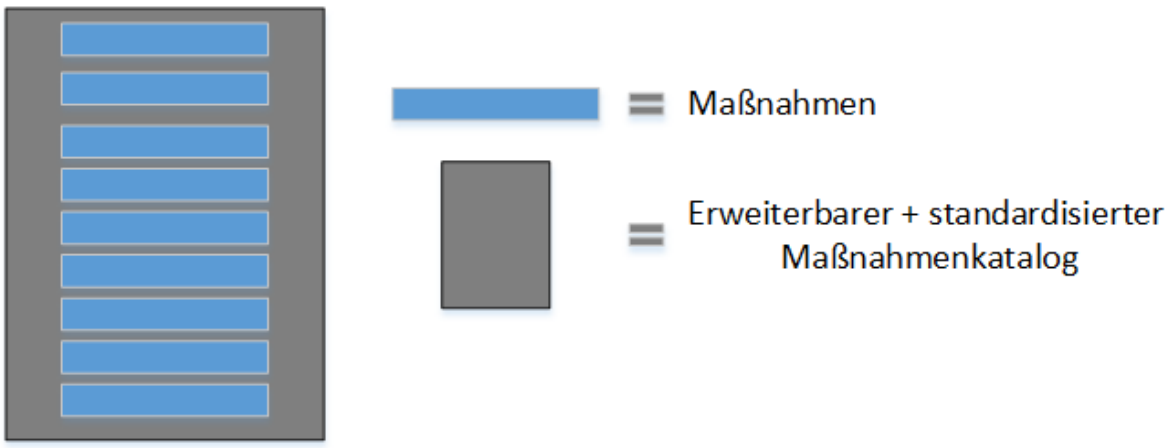


Bild 7.2: Veranschaulichung des Maßnahmenkatalogs

3 Code Scanning und Audit

4 Einbeziehung der Rechtsabteilung

5 Open Source Software Repositories

6 Einkaufsprozesse für Open Source Software

7 Organisation und Verantwortlichkeiten

8 Schulungen

9 Zuliefererbeziehung

Wie mit Hilfe dieses Maßnahmenkatalogs die Bewertung durchgeführt wird, wird in Abschnitt 7.3 im Allgemeinen und an Hand der neun Maßnahmen im Speziellen gezeigt.

7.2.4 Durchführungsverantwortlichkeit

Was weiterhin zu klären ist, ist die Frage der Verantwortlichkeit der Durchführung der Zertifizierung. Wer führt diese durch? Soll dies extern oder intern passieren (siehe Abbildung 7.3)? Außerdem ist es wichtig, ob die Zertifizierung in irgendeiner Weise auch verpflichtend sein kann.

Bild 7.3: Konflikt der internen beziehungsweise externen Zertifizierung

7.2.4.1 Externe Zertifizierung

Eine externe Zertifizierung bietet das objektivste Ergebnis. Allen Teilnehmern in der Lieferkette sollte an einem objektiven Ergebnis gelegen sein, da dies die Transparenz fördert. Dazu muss eine externe Kontrollinstanz eingerichtet werden, die Unternehmen einer Prüfung unterzieht und den Prozess, der in Abschnitt 7.4 genannt wird, durchführt. Eine Grundlage dazu ist ein standardisierter Maßnahmenkatalog. Dieser Standard muss mindestens für die Dauer einer Zertifizierung für eine Lieferkette gleich bleiben, damit für

jedes Unternehmen in einer Lieferkette die gleiche Bewertungsgrundlage herangezogen wird.

7.2.4.2 Interne Zertifizierung

Bei einer internen Zertifizierung werden im Vergleich zur externen alle Aktionen der Zertifizierung innerhalb des Unternehmens durchgeführt und keine externe Instanz in den Prozess eingebunden. Es sollte dennoch eine standardisierte Fassung des Maßnahmenkatalogs für eine Lieferkette vorliegen, um auch in diesem Fall eine gleiche Bewertungsgrundlage sicherzustellen. Um ein Mindestmaß an Objektivität zu gewährleisten, sollten alle Schritte, Annahmen und Ergebnisse, die im Laufe der Zertifizierung entstehen, umfassend dokumentiert werden.

7.2.4.3 Verpflichtende Zertifizierung

Es stellt sich zudem die Frage, ob eine Zertifizierung innerhalb einer Lieferkette verpflichtend oder nicht verpflichtend gestaltet werden sollte. Die Frage lässt sich dahingehend auf zwei verschiedene Weisen beantworten.
Wenn die Zertifizierung nicht verpflichtend, sondern freiwillig gehalten werden soll, dann würde auf Grund von Selbstregulierung und Wunsch nach sicherer Verwendung von Open Source Software, nur für diejenigen Zulieferer Platz in der Lieferkette sein, die sich einer Zertifizierung unterzogen haben. Denn es wäre nur auf Grund von erheblicher finanzieller Einsparung erklärbar, weshalb man sich als Unternehmen für einen nicht-zertifizierten Zulieferer entscheidet und nicht für einen zertifizierten. Deshalb sollte sich jedes Unternehmen, das Teil einer Lieferkette sein will, einer Zertifizierung unterziehen.
Auf der anderen Seite sollte man sich fragen, warum sich ein Unternehmen keiner Zertifizierung freiwillig unterziehen will. Ist sich das Unternehmen seiner Risiken bewusst? Weiß das Unternehmen über etwaige Verletzungen Bescheid? Natürlich könnte ein nicht-zertifiziertes Unternehmen auch einen geeigneten Schutz vor den Risiken implementiert haben, aber warum sollte man dies nicht der Öffentlichkeit zeigen?
Deshalb sollte die Antwort auf die Frage der Verpflichtung lauten, dass es eigentlich nicht relevant ist, ob die Pflicht vorliegt oder nicht, denn es ist im beidseitigen Interesse, dass eine Zertifizierung etabliert wird.

7.3 Klassifizierung in unterschiedliche Level

Wie im vorangegangen Abschnitt beschrieben, gibt es einen Maßnahmenkatalog, der verschiedene Vorkehrungen enthält. Diese Regelungen werden als Bewertungsgrundlage für die Qualität der Kontrolle der Open Source Risiken herangezogen. Sprich: Je mehr Maßnahmen im Unternehmen umgesetzt sind, umso besser ist das Unternehmen vor den Risiken geschützt. Grundlegend stellt der Maßnahmenkatalog eine Art Checkliste dar. Jede erfüllte Maßnahme ist demnach ein Pluspunkt. Je mehr Pluspunkte, desto besser der Schutz vor Risiken.

7.3.1 Klassifizierung mit Hilfe des Maßnahmenkatalogs

Dies soll an dieser Stelle differenzierter betrachtet werden. Es ist keineswegs so zu verstehen, dass die Maßnahmen als flache ungeordnete Liste vorliegen. Vielmehr unterliegt der Katalog einer Priorisierung und einer Gewichtung. Diese Gewichtung erfolgt auf Grund der Wichtigkeit und Wirksamkeit der Maßnahme in Hinblick auf das angestrebte Label.

7.3.1.1 Gewichtung der Maßnahmen

So sind bestimmte Maßnahmen für Label 1 wichtiger als für Maßnahme 2, wiederum andere Maßnahmen sind am wichtigsten für Label 2 und weniger wichtig für Label 1 und 3. Dem wird durch die unterschiedliche Gewichtung je Label Rechnung getragen. Es gibt drei verschiedene Gewichtungsstufen. Eine überaus wichtige Maßnahme wird mit dem Faktor 3 gewichtet, eine wichtige mit dem Faktor 2 und normale Maßnahmen mit dem Faktor 1. So kann es sein, dass eine sehr wichtige Maßnahme in Label 1 die Gewichtung 3 bekommt in Label 2 jedoch nur die Gewichtung 1 (siehe Abbildung 7.4). Diese Gewichtung geht mit der Aufnahme einer neuen Maßnahme in den Katalog einher. Gleichzeitig wird die unterschiedliche Gewichtung innerhalb jedes Labels aktualisiert. Dieser Maßnahmenkatalog sollte sich jedoch, wie oben bereits beschrieben, während eines Zertifizierungsvorgangs in einer Lieferkette nicht mehr ändern, um eine Vergleichbarkeit zu gewährleisten.

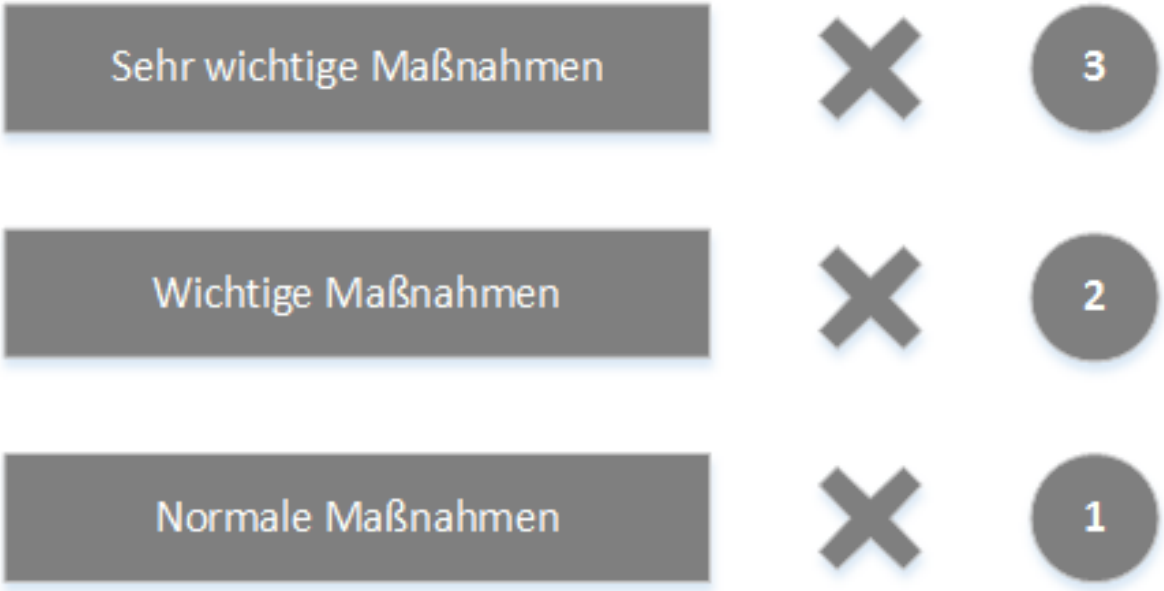


Bild 7.4: Veranschaulichung der Gewichtung

7.3.1.2 Errechnung des Erfüllungsgrades

Was hat die Gewichtung für einen Sinn? Wie oben bereits beschrieben, ist das Erfüllen einer Maßnahme ein Pluspunkt. Durch die Berücksichtigung der Gewichtung wird dem Erfüllen einer höher gewichteten Maßnahme größere Bedeutung beigemessen. Demnach werden mit dem Implementieren einer mit Faktor 3 gewichteten Maßnahme 3 Pluspunkte berücksichtigt.
Diese Pluspunkte werden summiert und anschließend durch die Anzahl der möglichen Gesamtpunkte geteilt. Daraus errechnet sich eine Prozentzahl der erreichten und umgesetzten Maßnahmen und damit der Erfüllungsgrad. Dieser wird als Maßzahl für den Schutz vor Risiken definiert.
Zur Verdeutlichung werden drei Intervalle des Erfüllungsgrades angegeben:

Erfüllungsgrad 0-50%: schlechter Schutz vor den Risiken

Erfüllungsgrad 51-74%: mittlerer Schutz vor den Risiken

Erfüllungsgrad 75-100%: guter Schutz vor den Risiken

Diese Intervallgrenzen können jedoch auch noch weiter verfeinert werden, um eine genauere Unterscheidung zu ermöglichen. Im Rahmen dieser Arbeit ist die Unterteilung in drei Intervalle jedoch ausreichend.
Der Erreichungsgrad wird bei jedem Zertifikat mit angegeben und ermöglicht somit die

transparente Bewertung des zertifizierten Unternehmens.

7.3.1.3 Beispielhafte Rechnung an Hand der vorgestellten Maßnahmen

An dieser Stelle soll eine beispielhafte Errechnung auf Grund der oben angegebenen neun Maßnahmen gezeigt werden. Zuerst wird die Gewichtung des Maßnahmenkatalogs hinsichtlich der Label angegeben:

Diese Gewichtung erhebt nicht den Anspruch die einzig mögliche Gewichtung zu

Label/Gewichtung	Label 1	Label 2	Label 3
Faktor 3	4,8,3	3,8,7	3,8,7
Faktor 2	7,9,2	4,2,5	4,2,5
Faktor 1	1,6,5	1,6,9	1,6,9

Tabelle 7.1: Beispielgewichtung des Maßnahmenkatalogs[1]

sein. Es ist jedoch wichtig der Tatsache Rechnung zu tragen, dass Label 1 auf die Einbindung von Open Source Software verzichtet. Deshalb ist die Maßnahme des Open Source Repositories weitaus weniger wichtig als es für die beiden anderen Label ist.
Bei dieser Beispielgewichtung würde es genügen, dass die drei sehr wichtigen Maßnahmen erfüllt sind, um einen Erfüllungsgrad von 50% zu erreichen. Jede Maßnahme, die zusätzlich erfüllt ist, würde den Erfüllungsgrad weiter in die Höhe treiben. Damit wäre die Wichtigkeit der drei wichtigsten Maßnahmen unterstrichen und hervorgehoben.

7.3.2 Optische Ausprägungen der Label

Doch wie kann ein Kunde eines Zulieferers schnell und transparent feststellen, welches Zertifikat und welchen Erfüllungsgrad ein Zulieferer besitzt? Und noch viel wichtiger: Wie kann auf einen Blick und transparent das Zertifikat und der Erfüllungsgrad festgestellt werden?
Dies wird erreicht, indem man für jedes Label ein optisches Äquivalent einführt. Dabei spiegelt die Zahl in der Mitte des Kreises die Art des Labels wieder und die Farbe der Kreisfüllung den Erfüllungsgrad. Dabei spiegelt grün den guten Schutz, gelb den normalen Schutz und rot den schlechten Schutz entsprechend den drei Intervallgrenzen für den Erfüllungsgrad wider (siehe Abbildungen 7.5 und 7.6). Hierdurch wird es ermöglicht, dass

ein Unternehmen nur durch die optische Ausprägung des Zertifikats bereits als potentiell risikoarmes oder potentiell risikoreiches Unternehmen identifiziert werden kann.
Zusätzlich zu den optischen Grundmerkmalen sollen Spezialmerkmale besondere Eigen-

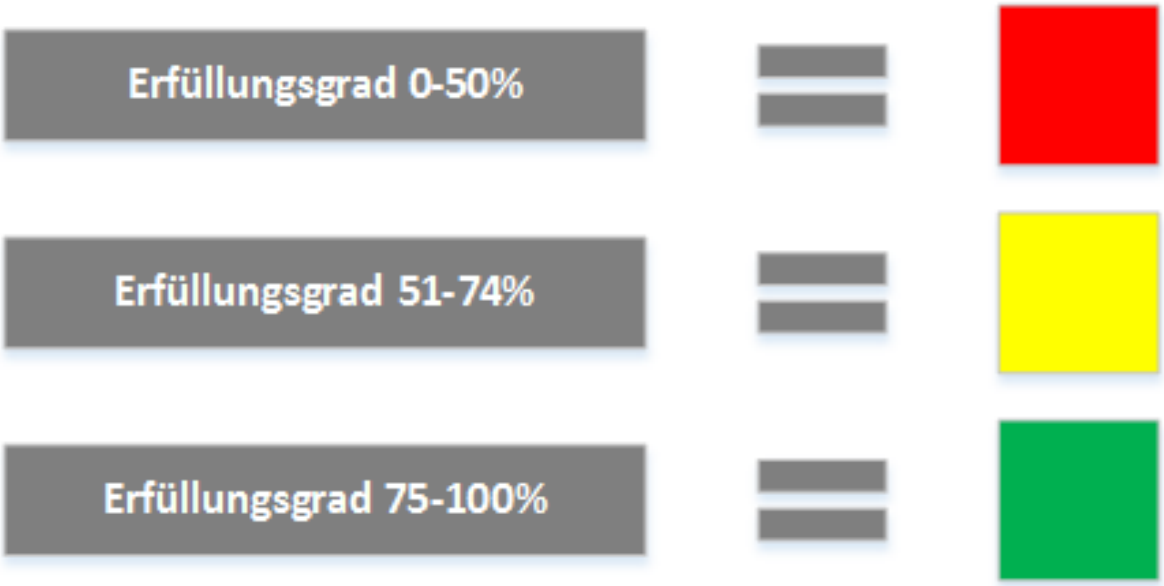


Bild 7.5: Zuordnung der Farben zum Erfüllungsgrad

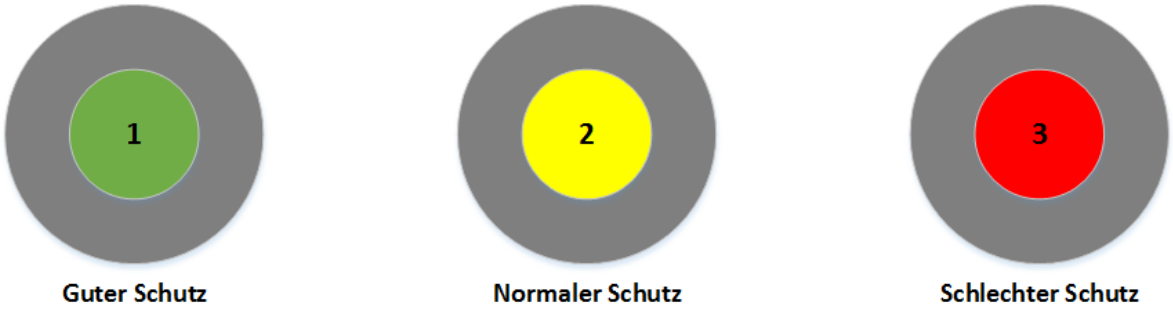


Bild 7.6: Optische Ausprägung der Label

schaften des Unternehmens unterstreichen. Dies können zum Beispiel sein (siehe Bild 7.7):

- Sternsymbol für 100% Erfüllungsgrad
- großes zusätzliches *G* zur Kennzeichnung der Benutzung von GPL-Lizenz
- Tendenzangabe mit Pfeilen bei wiederholter Zertifikatsprüfung

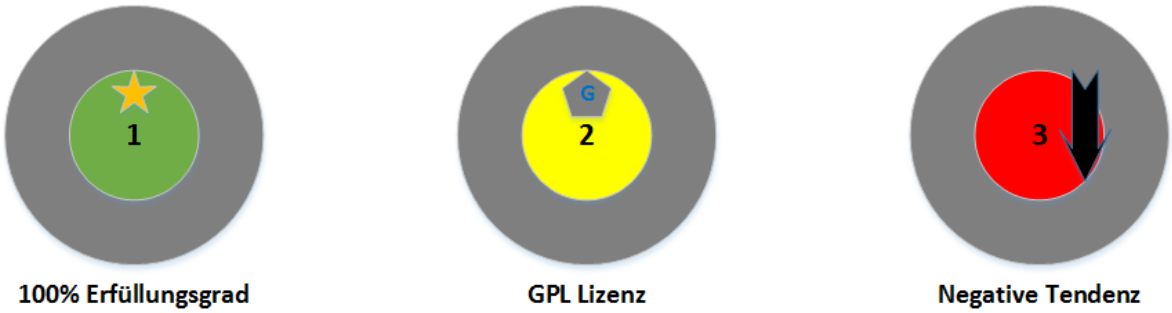


Bild 7.7: Optische Ausprägung der Label mit Sonderzeichen

7.4 Prozess

Egal ob die Zertifizierung schlussendlich verpflichtend oder freiwillig umgesetzt wird, egal ob die Zertifizierung extern oder intern durchlaufen wird und egal welcher Erfüllungsgrad am Ende als Ergebnis vorliegt. Es wird ein Prozess benötigt, der vorgibt, welche Aktivitäten wann und in welcher Reihenfolge von wem durchzuführen sind, um ein Zertifikat zu erlangen. Dieser besteht aus fünf Schritten:

1 Definition des zu erreichenden Labels

2 Code Analyse und Audit

3 Analyse des Unternehmens und der Open Source Maßnahmen

4 Erteilung des Labels mit entsprechendem Erfüllungsgrad

5 Aufzeigen von Verbesserungen

Diese fünf Prozessschritte werden nachfolgend näher erläutert. In Abbildung 7.8 wird der Prozess veranschaulicht.

7.4.1 Definition des zu erreichenden Labels

Im ersten Schritt des Prozesses wird das Label definiert, für das das Unternehmen eine Zertifizierung wünscht. Damit wird für das Audit festgelegt, welcher Maßnahmenkatalog zur Anwendung kommt. Die Verantwortlichkeiten sind dabei so verteilt, dass die Definition des Labels vom Unternehmen selbst zu erfolgen hat.

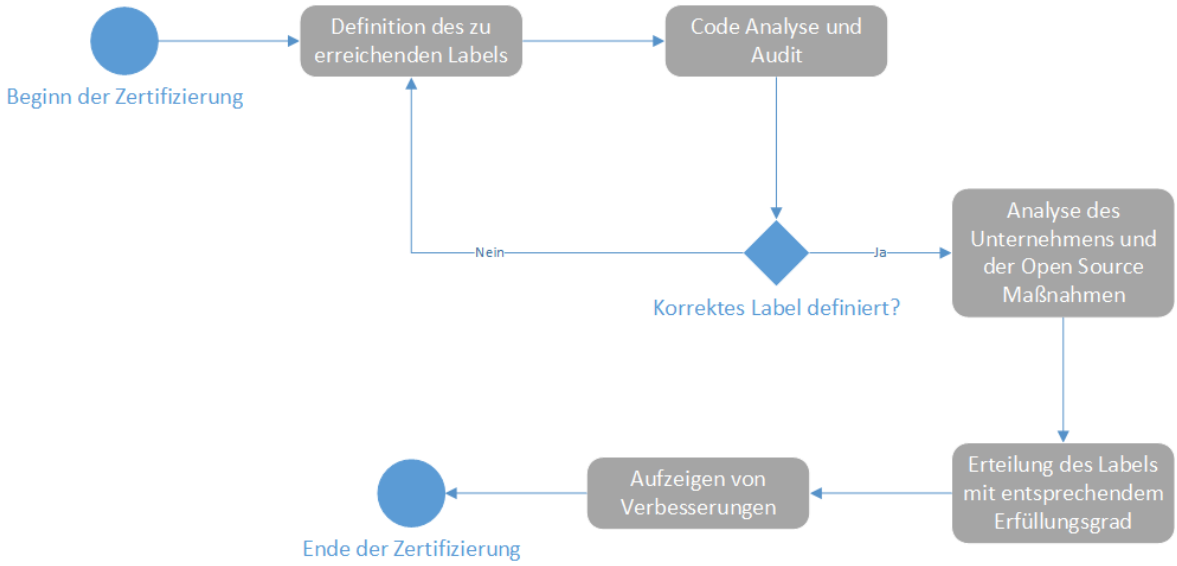


Bild 7.8: Darstellung des Zertifizierungsprozesses

7.4.2 Code Analyse und Audit

Im zweiten Prozessschritt wird entsprechend des zur Zertifizierung festgelegten Labels eine Analyse des Codes des Unternehmens durchgeführt. Dies hat den Hintergrund, dass zum Beispiel ein Unternehmen, das sich um eine Label 1 Zertifikat bemüht bereits unwissentlich Open Source Code integriert. Falls dies der Fall ist, muss der erste Prozessschritt mit einer geänderten Definition des angestrebten Labels erfolgen. Die Verantwortlichkeit liegt dabei beim Zertifizierungsverantwortlichen. Dieser kann entweder extern als auch intern sein, je nachdem welche Verfahrensweise diesbezüglich vereinbart wurde.

7.4.3 Analyse des Unternehmens und der Open Source Maßnahmen

Der dritte Schritt des Prozesses beinhaltet die eigentliche Analyse des Unternehmens. Dabei werden die durchgeführten Praktiken, Prozesse und die implementierten organisatorischen Verfahrensweisen begutachtet und hinsichtlich des Maßnahmenkatalogs analysiert. Der Maßnahmenkatalog muss sich nach dem in der speziellen Lieferkette vereinbarten Standard richten, damit Vergleichbarkeit gegeben ist. Die Verantwortlichkeit hat der Zertifizierungsverantwortliche. Ihm sind diesbezüglich auch Einsicht in alle

Unternehmensteile zu gewähren, um eine komplette Analyse zu ermöglichen und vollständigen Einblick in die Praktiken zu bekommen.

7.4.4 Erteilung des Labels mit entsprechendem Erfüllungsgrad

In dieser Stufe des Prozesses werden die im dritten Schritt herausgearbeiteten Maßnahmen des Maßnahmenkatalogs bewertet und aufsummiert. Daraus wird entsprechend der zuvor festgelegten Bewertungsgrundlage und Gewichtung der Erfüllungsgrad errechnet. Anschließend wird dem Unternehmen das Zertifikat überreicht beziehungsweise die Erlaubnis erteilt dieses in Verbindung mit dem Unternehmen zu führen. Dieses Zertifikat ist bindend und darf vom Unternehmen auch nicht verändert werden. Die Verantwortung der Errechnung und die Kompetenz des Erteilens obliegt allein dem Zertifikatsverantwortlichen.

7.4.5 Aufzeigen von Verbesserungen

Im letzten Prozessschritt wird dem zertifizierten Unternehmen ein umfassender Bericht über die Analyse des Unternehmens zur Verfügung gestellt. Dabei steht das Aufzeigen von Schwachstellen für den Unternehmensverantwortlichen im Vordergrund. Mit Hilfe dieser aufgezeigten Lücken im Schutz soll der Unternehmensverantwortliche zur Verbesserung des Schutzes animiert werden und der Spielraum für Verbesserung aufgezeigt werden.

7.5 Integration in die Lieferkette

Der gerade beschriebene Prozess sollte in einer kompletten Lieferkette Anwendung finden. Dabei ist es wichtig, dass der Prozess für diese Lieferkette standardisiert und identisch abläuft. Wenn Maßnahmenkatalog und Prozess standardisiert sind und jeder Zulieferer und zusätzlich jeder potentielle Zulieferer die Zertifizierung durchführt, dann besteht eine Transparenz hinsichtlich des Schutzes vor den Risiken von Open Source Software in der Lieferkette.
Doch was ist, wenn sich über die Zeit noch komplexere und abgeänderte Risiken ergeben, die durch die bisher begutachteten Maßnahmen nicht abgedeckt sind? Um diesem Umstand entgegen zu wirken, sollte vereinbart werden, dass das Label nach einer

festgelegten Zeitspanne seine Gültigkeit verliert und eine erneute Zertifizierung notwendig wird. Dies kann entweder vertraglich festgeschrieben werden oder auf freiwilliger Basis geschehen. Es kann außerdem ein Mindesterfüllungsgrad innerhalb der Lieferkette fixiert werden, der vorgibt, welcher Erfüllungsgrad erreicht sein muss, um Teil der Lieferkette zu sein. Falls bei erneuter Zertifizierung eines bereits an der Lieferkette teilnehmenden Zulieferers eine Unterschreitung des festgelegten Erfüllungsgrad attestiert wird, dann bekommt dieser eine Übergangsfrist eingeräumt Anpassungen zu machen, bevor eine erneute Zertifizierung durchgeführt wird.
Das Konzept des Schutzes vor den Risiken von Open Source in der Lieferkette ist, ähnlich den Compliance Anforderungen von Unternehmen, ein dynamisches Feld, da sich die Rahmenbedingungen schnell ändern können. Daher ist es wichtig, dass der Maßnahmenkatalog bei Änderung der Anforderung angepasst und gegebenenfalls erweitert wird und dieser bei der verpflichtenden erneuten Zertifizierung Anwendung findet. Damit wird ein dynamischer Kreislauf aus Definition von Maßnahmen, Implementierung dieser Maßnahmen, Zertifizierung und anschließender kontinuierlicher Verbesserung, umgesetzt.

7.5.1 Einsetzung einer Standardisierungsinstanz

Als Alternative zu dem bisher genannten kann eine Institution eingerichtet werden, die einen Standard für den Maßnahmenkatalog definiert, und die Einhaltung dieses Standards durchsetzt. Außerdem werden von dieser Institution Regeln und Richtlinien definiert, wann und unter welchen Umständen die Gültigkeit des Label, ausläuft. Diese Institution kann sowohl aus dem akademischen Umfeld kommen, als auch eine eigenständige nicht-kommerzielle Gesellschaft sein. Es sollte in diesem Falle jedoch besonders auf Objektivität geachtet werden.
Der große Vorteil von einer unabhängigen Institution ist die noch größere Vergleichbarkeit auch über viele Lieferketten hinweg. Außerdem ist damit gewährleistet, dass das Verfahren der Zertifizierung und die Richtlinien, die damit einhergehen, kontinuierlich und in gleicher Weise verbessert und weiterentwickelt werden.

7.5.2 Beteiligung an mehreren Lieferketten

Wie wird verfahren, wenn ein Unternehmen in mehreren Lieferketten beteiligt ist? Dies ist ein häufig vorkommender Fall, und es bedarf einer detaillierten Analyse.

Im ersten Falle, in der es keine Standardisierungsinstitution gibt, ist dafür Sorge zu tragen, dass die beiden Zertifizierungsprozesse und Maßnahmenkataloge miteinander vereinbar sind. Das Anstreben der stringenteren Zertifizierungsanforderungen ist zu empfehlen, um in jeder Lieferkette den größtmöglichen Schutz zu gewährleisten.
Im Falle einer Standardisierungsinstitution wird in allen Lieferketten der gleiche Maßnahmenkatalog und der gleiche einheitliche Prozess zur Anwendung gebracht. Dadurch entstehen keinerlei zusätzliche beachtenswerte Unterschiede. Die Label sind über die Grenzen der einzelnen Lieferkette hinweg vergleichbar.

7.5.3 Auswirkungen und Folgen auf die Lieferkette

Damit lassen sich folgende Auswirkungen und Folgen in der Lieferkette beobachten:

Transparenz Die Transparenz wird erreicht durch das Aufdecken der Maßnahmen, die jedes Unternehmen zum Schutz gegen die Risiken von Open Source Verwendung implementiert. Diese Maßnahmen würden ansonsten unter Verschluss bleiben. Speziell durch die optischen Ausprägung des Labels ist es möglich, einen schnellen und transparenten Überblick über die Unternehmenspraxis bezüglich Open Source Verwendung zu erhalten.

Selbstreflexivität Durch das Aushändigen eines Berichtes durch den Zertifizierungsverantwortlichen wird das Unternehmen befähigt, seine eigene Position am Markt zu erkennen. Jedes Unternehmen weiß demnach wo es sich befindet und kann sich über die getroffenen Maßnahmen und Entscheidungen hinsichtlich von Open Source Verwendung Gedanken machen.

Motivation zur Verbesserung des Schutzes Nach der Selbstreflexivität, die jedes Unternehmen befähigt sich Gedanken über die eigene Position zu machen, folgt die Motivation sich zu verbessern, um die Marktposition zu stärken. Der Bericht des Zertifizierungsverantwortlichen zeigt mit Hilfe des Maßnahmenkatalogs die Lücken im Schutz auf und befähigt damit die Unternehmen sich dieser Schwachstellen anzunehmen und den Schutz zu stärken. Motivation kann in dieser Hinsicht sowohl die Vermeidung von Risiken und den damit verbundenen Schäden des Unternehmens sein, als auch finanzielle Anreize durch das Etablieren in neuen Lieferketten.

Verminderung von Risiken bezüglich Open Source Es ist klar ersichtlich, dass wenn alle Unternehmen sich in einer Lieferkette an einer derartigen Zertifizierung

beteiligen und ein vertraglicher Mindesterfüllungsgrad fixiert wird, die Risiken durch Open Source effektiv vermindert werden können.

Einfachheit und Verständlichkeit Durch den Maßnahmenkatalog und die optischen Ausprägungen, die nur wenige aber aussagekräftige optische Merkmale besitzen, ist es den an der Lieferkette beteiligten Unternehmen möglich, einfach und verständlich Einblick in die Open Source Vorkehrungen der anderen Unternehmen zu erhalten.

Verstärkung des Vertrauens in Lieferanten Vertrauen ist gut, Kontrolle ist besser. Durch die Zertifizierung und der kontinuierlichen Verbesserung eines Unternehmens bezüglich des Erfüllungsgrades wird das Vertrauen in Lieferanten gestärkt, da dies ein Zeichen dafür ist auch in anderen Unternehmensbereichen kontinuierliche Verbesserungen einzuführen.

Einfachere Auswahl von Zulieferern Die Auswahl von Lieferanten wird derart vereinfacht, dass Unternehmen durch das erteilte Zertifikatslabel ein zusätzliches objektives Kriterium zur Hand hat, das Aufschluss über den Lieferanten bringt. Es ist jedoch wichtig zu erwähnen, dass das Label nicht als alleiniges Merkmal zur Auswahl des Lieferanten herangezogen werden kann. Denn es kann durchaus vorkommen, dass sich das Unternehmen für einen Lieferanten mit einem schlechten Erfüllungsgrad entscheidet, dieses Unternehmen sich in diesem Fall jedoch des Risikos bewusst ist.

7.6 Fallbeispiel

In diesem Abschnitt werden die drei eingeführten Unternehmen erneut betrachtet. Dabei wird auf die Zertifizierung eingegangen und wie sich der Prozess des in diesem Kapitel eingeführten Modells für diese Unternehmen darstellt.

7.6.1 Beispielszenario 1: PropForSuccess GmbH

Da die PropForSuccess GmbH ein Unternehmen ist, das keinerlei Open Source Software implementiert, definiert die Unternehmensleitung für sich, dass man ein Label 1 anstrebt. Des Weiteren ist PropForSuccess GmbH Teil einer Lieferkette, die einen vertraglich fixierten minimalen Erfüllungsgrad von 50% angegeben hat, und die Zertifizierung wird von einem externen Zertifizierungsinstitut ZiTo abgewickelt.
Im zweiten Prozessschritt stellt ZiTo fest, dass sich Teile von Open Source Code im

Unternehmen befinden. Nun hat PropForSuccess GmbH die Wahl: Entweder man definiert sich Label 2 als neue Zielsetzung oder man versucht den entdeckten Open Source Code zu beseitigen und behält die Zielsetzung Label 1 bei. Außerdem fühlt man sich durch diesen Fund bestätigt, dass eine Zertifizierung und der damit verbundenen Sichtbarmachung der Maßnahmen und Sicherheitslücken auch für einen proprietären Entwickler Sinn macht. Man hat sich dazu entschlossen weiterhin auf das Label 1 zu setzen. ZiTo bescheinigt nach der Analyse des Unternehmens einen Erfüllungsgrad von 70%. Wie sich später herausstellt, hat dieser hohe Erfüllungsgrad der PropForSuccess GmbH ermöglicht weitere Kunden zu gewinnen, für die sie als Zulieferer tätig waren.
Dadurch war die Zertifizierung für PropForSuccess GmbH aus zweierlei Gründen ein Erfolg: Die Risiken wurden vermindert und gleichzeitig waren finanzielle Gewinne zu verzeichnen.

7.6.2 Beispielszenario 2: MixIT GmbH

Im Gegensatz zu PropForSuccess GmbH implementiert die MixIT GmbH auch Open Source Code und definiert das Label 2 für sich als Zielsetzung zur Zertifizierung. Da auch in der Lieferkette, in der die MixIT GmbH beteiligt ist, ein Mindesterfüllungsgrad vertraglich fixiert ist, muss das Unternehmen 60% Abdeckung der Vorkehrungen aus dem Maßnahmenkatalog erfüllen. Außerdem ist eine Wiederholung der Zertifizierung alle 2 Jahre festgelegt worden.
Nachdem der Zertifizierungsverantwortliche die Zertifizierung durchgeführt hat, wurden der MixIT GmbH ein Erfüllungsgrad von genau 60% attestiert. Da dies ausreichend für das weitere Partizipieren an der Lieferkette ist, wurde das Ergebnis mit großer Freude entgegen genommen.
Nachdem weitere zwei Jahre vergangen sind und die MixIT GmbH viele wichtige Projekte abgewickelt hat, steht nun die wiederholte Zertifizierung mit einem erweiterten Maßnahmenkatalog an. Die MixIT GmbH hat jedoch keine Verbesserungen der implementierten Vorkehrungen veranlasst. Deshalb beträgt der Erfüllungsgrad der erneuten Zertifizierung nur noch 40%, was ein weiteres Teilnehmen in der Lieferkette nicht möglich macht. Die MixIT GmbH bekommt daraufhin ein halbes Jahr Übergangsfrist eingeräumt die Maßnahmen auszubauen.
Die MixIT GmbH hat daraus gelernt, dass die Risiken und Maßnahmen in Hinblick auf Open Source ein dynamisches Feld sind und kontinuierliche Anpassungen verlangt.

7.6.3 Beispielszenario 3: LibertatemVincIT GmbH

Die LibertatemVincIT GmbH entwickelt selbst Open Source Software und beantragt deswegen ein Label 3. Die Lieferkette, in der die LibertatemVincIT GmbH tätig ist, hat keine verpflichtende Zertifizierung und lässt eine interne Zertifizierung zu. Es wird jedoch gefordert, falls eine Zertifizierung erfolgt, eine umfangreiche Dokumentation anzulegen und an alle Unternehmen in der Lieferkette auszuhändigen ist.
Der Grund weshalb sich die LibertatemVincIT GmbH dazu entschlossen hat trotz der Freiwilligkeit der Zertifizierung ein Label zu beantragen, ist, dass sie ein Bild von den eigenen getroffenen Maßnahmen bekommt. Die Hoffnung ist, dass man damit die Risiken durch Aufzeigen der weiter bestehenden Schwachstellen und das Schließen dieser vermindert.
Dass man bisher im Unternehmen das Thema der Gefahren von Open Source ernst genommen hat, sieht man am Erfüllungsgrad, der nach der umfangreichen Analyse 100% beträgt. Damit ist für jedes andere Unternehmen der Prozess von LibertatemVincIT GmbH transparent nachvollziehbar. Jeder andere Lieferant kann sich damit sicher sein, dass LibertatemVincIT GmbH alles tut um die Open Source Risiken zu minimieren.

7.7 Zusammenfassung

In diesem Kapitel wurde ein Vorschlag für ein Zertifizierungsmodell in der Open Source Lieferkette gegeben. Das große Ziel dieses Modell ist es, die Risiken, die in Verbindung mit Open Source Software entstehen können, abgemindert werden und bestenfalls gänzlich kontrollierbar gemacht werden können. Dazu sind von Unternehmensseite bestimmte Vorkehrungen zu treffen, die zur Reduzierung der Schwachstellen und Gefahren geeignet sind. Diese Vorkehrungen werden im Konzept des Maßnahmenkatalogs zusammengefasst und aufgeführt. Dieser gewichtete Maßnahmenkatalog dient als Grundlage zur Bewertung der Unternehmen.
Die Bewertung erfolgt zweidimensional. Die eine Dimension bildet das Label mit drei möglichen Ausprägungen: kein Open Source Einsatz, Open Source Integration, eigene Open Source Entwicklung. Die zweite Dimension ist der Erfüllungsgrad. Dieser wird in drei Intervallen bewertet und ist Gradmesser für den Umfang der Implementierung der Maßnahmen. Das Label inklusive Erfüllungsgrad wird als optisches Zertifikat zur Verfügung gestellt.

Zur Zertifizierung wird ein Prozess durchlaufen. Dieser Prozess beinhaltet eine Analyse des Unternehmens und hat die zweidimensionale Bewertung als Ergebnis. Wie oben bereits erwähnt ist es wichtig, dass die gesamte Lieferkette diesen Prozess durchführt, um eine mögliche Schwachstelle in der Lieferkette und den damit verbundenen Auswirkungen auf jeden einzelnen Agenten in der Lieferkette zu eliminieren.
Durch das Einsetzen einer Standardisierungsinstanz ist das Verwenden der Label über die Grenzen einer Lieferkette hinweg gewährleistet. Diese überwacht den Maßnahmenkatalog und den Prozess.
Zum Abschluss des Kapitels wurden in Abschnitt 7.5 die Auswirkungen und positiven Effekte des Modells auf die Lieferkette besprochen und dargestellt.

8 Zusammenfassung und Ausblick

Dieses Kapitel ist aufgeteilt in zwei Abschnitte. Der erste Abschnitt enthält eine Zusammenfassung der vorliegenden Arbeit. Der zweite Teil gibt einen Ausblick auf mögliche weiterführende Arbeiten und die Forschung.

8.1 Zusammenfassung

Die vorliegende Arbeit beginnt mit einer Erarbeitung der Grundlagen, die für das weitere Verständnis von Nöten sind. Diese sind zu finden im Kapitel 3. Dieses Kapitel spaltet sich in drei Hauptteile auf. Der erste Abschnitt führt in das Gebiet der Open Source Software ein. Dazu wird zu Beginn die Philosophie der Open Source Bewegung auf Grundlage der Open Source Definition angegeben. Anschließend werden die wichtigen Punkte der Lizenzierungsproblematik und des geistigen Eigentums vorgestellt. Diese beiden Punkte werden später in der Arbeit oft aufgegriffen, weshalb an dieser Stelle auf die risikofördernden Faktoren der beiden Thematiken verzichtet wird. Das Thema Open Source Einsatz im Unternehmen wird als Abschluss in diesem Teil des Kapitels betrachtet, da es im weiteren Verlauf der Arbeit explizit um dieses Einsatzgebiet von Open Source Software geht. Zusätzlich zu den Grundlagen der Open Source Software erstreckt sich die Grundlagenerarbeitung noch auf die Einführungen zu Softwarezertifizierungsmodellen, wie ISO 9001, CMM und SPICE und zu der Softwarelieferkette.
Das darauffolgende Kapitel 4 stellt vier Forschungsarbeiten vor, die eine ähnliche Thematik aufgreifen wie die vorliegende Arbeit. Es sei jedoch angemerkt, dass die Schwerpunkte der Arbeiten jeweils unterschiedlich sind und andere Herangehensweisen an die Problematik der Bewertung von Open Source Software darstellen.
Im Kapitel 5 werden die Risiken, die in Verbindung mit der Nutzung von Open Source Software im Unternehmen stehen, betrachtet. Es werden diesbezüglich drei Risikogebiete vorgestellt: Juristische Risiken, ökonomische Risiken und gesellschaftliche Risiken. Diese Gebiete bedingen sich gegenseitig und treten deshalb meist gemeinsam auf. Besonders problematisch ist vor allem der Umgang mit geistigem Eigentum im Allgemeinen und

Patenten, Copyright und Open Source Lizenzen im Speziellen. Durch falsche Benutzung dieser Konzepte entstehen Schäden für das Unternehmen, die bis zur Geschäftsaufgabe führen können. Aber auch wenn es nicht zum äußersten Fall der Geschäftsaufgabe kommt, sind die Auswirkungen der Risiken potentiell schwerwiegend. Es können schwere monetäre Einbußen, gesellschaftliche Ächtung, Verkaufsverbote oder sonstige Sanktionen entstehen. Um diesen Risiken entgegenzutreten haben viele Unternehmen best practices entwickelt. Diese werden in Kapitel 6 vorgestellt. Diese Maßnahmen erstrecken sich über das gesamte Unternehmen, um eine Gefährdung durch Open Source Software an keiner Stelle des Unternehmens zuzulassen. Die Implementierung eines solchen Maßnahmenpakets wird vielfach unter dem Begriff Open Source Policy zusammengefasst. Gemeint sind jedoch einzelne Vorkehrungen, die sich auf die Transparenz, die Verteilung von Kompetenz und Verantwortung, Vereinfachung und Standardisierung von Prozessen in Bezug auf Open Source Software abzielen. Die Folge sind für diese Unternehmen, dass die potentiellen Risiken durch den korrekten Umgang mit Open Source Software verhindert oder zumindest abgemildert werden können.
In der vorliegenden Arbeit werden jedoch nicht nur einzelne Unternehmen betrachtet. Es wird eine komplexe Softwarelieferkette betrachtet. Es können demnach durch inkorrekte Verfahrensweisen in einem Teil der Lieferkette Schäden für alle anderen Beteiligten führen. Deshalb ist es notwendig das Konzept der Open Source Policy um einen Zertifizierungsprozess zu erweitern, der das Umsetzen der best practices transparent überprüfbar für alle anderen Unternehmen in der Lieferkette macht.
Ein Vorschlag für ein solches Zertifizierungsmodell wird in Kapitel 7 angegeben. Als Grundlage für das Modell dient der Maßnahmenkatalog, der erweiterbar und standardisiert ist. Maßnahmen sind in diesem Zusammenhang die best practices aus dem vorherigen Kapitel. Als Ziel der Zertifizierung stehen drei unterschiedliche Label. Jeweils eines für Unternehmen, die Open Source aktiv nutzen, für Unternehmen, die auch selbst Open Source Software entwickeln, und Unternehmen, die weder Open Source nutzen noch selbst entwickeln. Als Kennzahl für die Güte für den Schutz vor den Risiken wird der Erfüllungsgrad definiert. Dieser wird durch die Implementierung der Maßnahmen aus dem Maßnahmenkatalog errechnet. Die Vorkehrungen darin sind gewichtet. Die Gewichtung kann sich von Label zu Label unterscheiden, um dem unterschiedlichen Stellenwert der Open Source Nutzung Rechnung zu tragen. Die Zertifizierung wird mit Hilfe eines Prozesses durchgeführt, der sich in fünf Teilschritte aufgliedert:

1. Definition des zu erreichenden Labels
2. Code Analyse und Audit

3. Analyse des Unternehmens und der Open Source Nutzung
4. Erteilung des Labels mit entsprechenden Erfüllungsgrad
5. Aufzeigen von Verbesserungen

Im Kapitel werden noch weitergehende Informationen zur Durchführungsverantwortlichkeit, der Notwendigkeit einer Wiederholung der Zertifizierung, einer Etablierung einer Standardisierungsinstanz und der Beteiligung eines Unternehmens an mehreren Lieferketten angegeben. Die schlichte Folge der Zertifizierung ist: Verminderung der Risiken durch Open Source Nutzung im Unternehmen - auch in komplexen Lieferketten.

8.2 Ausblick

Nachdem in der vorliegenden Arbeit ein Vorschlag für ein Zertifizierungsmodell für die Open Source Nutzung in der Softwarelieferkette aufgezeigt wurde, ist es von großem Interesse, dieses Modell in die Praxis zu überführen. Dabei sollte ein großer Fokus auf die Formulierung des Maßnahmenkatalogs gelegt werden, denn dieser dient als Grundlage für die Bewertung der Unternehmen. Es sollte deshalb ein Review der hier eingeführten Vorkehrungen durchgeführt werden. Dabei sollte beurteilt werden, ob all diese Maßnahmen nötig sind oder ob weitere Maßnahmen als Ergänzung sinnvoll sind. Außerdem sind für jede Vorkehrung Akzeptanzkriterien zu definieren, die eine Überprüfung der Implementierung erleichtern. Das Ergebnis ist ein standardisierter Maßnahmenkatalog, der als Grundlage für alle weiteren Zertifizierungen dienen soll. Zusätzlich wird dieser Katalog in Zukunft Veränderungen erfahren. Diese Veränderungen sollten iterativ in neuen Versionen des Katalogs eingearbeitet werden und daraufhin eine Neu-Zertifizierung der Unternehmen mit einer Übergangsfrist durchgeführt werden. Damit werden den immer komplexer werdenden Einsatzfeldern und Rahmenbedingungen von Open Source Software Rechnung getragen.
Zusätzlich zur praktischen Umsetzung sollte eine Untersuchung der Wirksamkeit des Modells durchgeführt werden. Damit lassen sich Aussagen in Hinblick auf die vermiedenen Risiken und der damit verbundenen Einsparungen der Unternehmen treffen. Denn es ist essentiell, dass die abgewendeten Schäden für das Unternehmen schwerer wiegen als die Kosten der Zertifizierung.

Literaturverzeichnis

[All05] Business Software Alliance. Open source and commercial software an in-depth analysis of the. Technical report, Business Software Alliance, 2005.

[Bad10] Virginia Badenhope. Pulling it all together: An open source action plan for in-house counsel, March 2010.

[BB02] Alan W Brown and Grady Booch. Reusing open-source software and practices: The impact of open-source on commercial vendors. In *Software Reuse: Methods, Techniques, and Tools*, pages 123–136. Springer, 2002.

[BFM07] Steffen Burger, Felix Friedrich, and Milan Mehner. Gnu general public license version 3. *Informatik & Gesellschaft*, 1:1, 2007.

[BK96] Adriana Bicego and Pasi Kuvaja. Software process maturity and certification. *Journal of Systems Architecture*, 42(8):611–620, 1996.

[Bla12a] BlackDuck. A breackthrough in software supply chain communications know your code, label your code and let others know your code. Technical report, BlackDuck, 2012.

[Bla12b] BlackDuck. Four steps to creating an effective open source policy. Technical report, BlackDuck, 2012.

[Bre04] Pearl Brereton. The software customer/supplier relationship. *Commun. ACM*, 47(2):77–81, February 2004.

[CF02] Reidar Conradi and Alfonso Fuggetta. Improving software process improvement. *IEEE Software*, 19(4):92–99, 2002.

[DG01] Jamie Dinkelacker and Pankaj K. Garg. Corporate source: Applying open source concepts to a corporate environment. In *1st Workshop on Open Source Software Engineering at ICSE 2001*, 2001.

[DM05] Linus Dahlander and Mats G Magnusson. Relationships between open source software companies and communities: Observations from nordic firms. *Research Policy*, 34(4):481–493, 2005.

[Duc13] Black Duck. Black duck open source strategy. Internet, August 2013.

[Egg06] Daniel Egger. Open source software ip risk audits: The emerging due diligence standard for technology m&a transactions. Technical report, Open Source Risk Management, 2006.

[Eil08] Siep Eilander. The acquisition of (open-source) software a guide for ict buyers in the public and semi-public sectors. Technical report, Goverment of Netherlands, 2008.

[ELF04] David S. Evans and Anne Layne-Farrar. Software patents and open source: The battle over intellectual property rights. *Virginia Journal of Law & Technology*, 9:1, 2004.

[Ell11] John Ellis. Open source compliance in the supply chain. Technical report, BlackDuck, 2011.

[EvS00] Mariki M. Eloff and Sebastiaan H. von Solms. Information security management: An approach to combine process certification and product evaluation. *Computers & Security*, 19(8):698–709, 2000.

[FAK04] Brian Fan, Andrew Aitken, and John Koenig. Open source intellectual property and licensing compliance: A survey and analysis of industry best practices, 2004.

[FF99] B. Farbey and A. Finkelstein. Exploiting software supply chain business architecture: a research agenda. In *1st Workshop on Economics-Driven Software Engineering Research (EDSER-1), 21st International Conference on Software Engineering*, 1999.

[FFM11] Fabrizio Fabbrini, Mario Fusani, and Eda Marchetti. Process scenarios in open source software certification. *ECEASST*, 48, 2011.

[FGL08] Andrea Fosfuri, Marco S Giarratana, and Alessandra Luzzi. The penguin has entered the building: The commercialization of open source software products. *Organization science*, 19(2):292–305, 2008.

[Fog05] Karl Fogel. *Producing open source software: How to run a successful free software project*. O'Reilly Media, Inc., 2005.

[Fou07] Free Software Foundation. Gnu general public license. Internet, Juni 2007.

[Fou13] The Linux Foundation. Spdx. Internet, August 2013.

[Fre01] C. Freericks. Open source standards on software process: a practical application. *Comm. Mag.*, 39(4):116–123, April 2001.

[FS05] Brian Fitzgerald and Nic Suzor. Legal issues for the use of free and open source software in government. *Melb. UL Rev.*, 29:412, 2005.

[GDE97] Craig M. Gustin, Patricia J. Daugherty, and Alexander E. Ellinger. Supplier selection decisions in systems/software purchases. *International Journal of Purchasing and Materials Management*, Fall:1, 1997.

[Gie12] Julie Giera. The costs and risks of open source software debunking the myths. Forrester Report, April 2012.

[Gro13] Open Source Risk Management Group. Open source risk management group homepage. Internet, August 2013.

[GS03] Jack Greenfield and Keith Short. Software factories: assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '03, pages 16–27, New York, NY, USA, 2003. ACM.

[Hah02] Robert W. Hahn. *Government Policy toward Open Source Software*. AEI-Brookings Joint Center for Regulatory Studies, 2002.

[Ham09] Jeffrey S Hammond. Best practices: Improve development effectiveness through strategic adoption of open source, 2009.

[Hec99] Frank Hecker. Setting up shop: The business of open-source software. *IEEE Software*, 16(1):45–51, January/February 1999.

[Hel01] Martin Helmreich. Best practices of adopting open source software in closed source software products. Master's thesis, FAU Erlangen-Nürnberg, 2001.

[HH11] Björn Hajek and Nick Holland. Making commercial use of open source. Technical report, Association of Corporate Counsen Europe, 2011.

[HP07] HP. Best practices in open source governance managing the selection and proliferation of open source software accross your enterprise. Technical report, HP, 2007.

[HPL05] Jesper Holck, Mogens Kühn Pedersen, and Michael Holm Larsen. Open source software acquisition: Beyond the business case. In Dieter Bartmann, Federico Rajola, Jannis Kallinikos, David E. Avison, Robert Winter, Phillip Ein-Dor, Jörg Becker, Freimut Bodendorf, and Christof Weinhardt, editors, *ECIS*, pages 1497–1508, 2005.

[HR] Martin Helmreich and Dirk Riehle. Geschäftsrisiken und governance von open-source in softwareprodukten.

[HR06] Wilhelm Hasselbring and Ralf Reussner. Toward trustworthy software systems. *IEEE Computer*, 39(4):91–92, 2006.

[Ini13] Open Source Initiative. Osi.org. Internet, 2013.

[JBF07] Slinger Jansen, Sjaak Brinkkemper, and Anthony Finkelstein. Providing transparency in the business of software: A modeling technique for software supply networks. In Luis M. Camarinha-Matos, Hamideh Afsarmanesh, Paulo Novais, and Cesar Analide, editors, *Virtual Enterprises and Collaborative Networks*, volume 243 of *IFIP Advances in Information and Communication Technology*, pages 677–686. Springer, 2007.

[Jon94] Capers Jones. Globalization of software supply and demand. *IEEE Software*, 11(6):17–24, 1994.

[Ken01] Dennis M Kennedy. Primer on open source licensing legal issues: Copyright, copyleft and copyfuture, a. . *Louis U. Pub. L. Rev.*, 20:345, 2001.

[KKS12] Harsha K Kalutarage, Padmanabhan Krishnan, and Siraj A Shaikh. A certification process for android applications. In *6th International Workshop on Foundations and Techniques for Open Source Software Certification (OpenCert 2012) held on 2nd October*, 2012.

[KM01] Bruce Kogut and Anca Metiu. Open-source software development and distributed innovation. *Oxford Review of Economic Policy*, 17(2):248–264, 2001.

[Kos07] Heli Koski. Oss production and licensing strategies of software firms. *Review of Economic Research on Copyright Issues*, 2:111–125, 2007.

[Lau08] Andrew M St Laurent. *Understanding open source and free software licensing*. O'Reilly Media, 2008.

[Lin12] Claudius Link. Patterns for the commercial use of open source: License patterns. In *European Conference on Pattern Languages of Programs (EuroPLoP) 2011(EuroPLoP) 2011 Workshop A*, volume 331, 2012.

[LT02] Josh Lerner and Jean Tirole. Some simple economics of open source. *The journal of industrial economics*, 50(2):197–234, 2002.

[Man06] Ronald J. Mann. Commercializing open source software: Do property rights still matter? *Harvard Journal of Law & Technology*, 20:1, 2006.

[MFS12] Pedro Martins, Joao P. Fernandes, and Joao Saraiva. A web portal for the certification of open source software. *OpenCert 2012*, 1:1, 2012.

[MJ01] Axel Metzger and Till Jaeger. Open source software and german copyright law. *IIC*, 32(1):52–73, 2001.

[MLP+01] John Morris, Gareth Lee, Kris Parker, Gary A. Bundell, and Chiou Peng Lam. Software component certification. *Computer*, 34(9):30–36, 2001.

[NRS96] Paul Nelson, William B. Richmond, and Abraham Seidmann. Two dimensions of software acquisition. *Commun. ACM*, 39(7):29–35, 1996.

[OFM+03] Siobhan O'Mahony, Fabrizio Ferraro, Mark Mortensen, Michael Schrage, and Rachel Campagna. Guarding the commons: How community managed software projects protect their work, 2003.

[O'H00] Fran O'Hara. European experiences with software process improvement. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 635–640. IEEE, 2000.

[Ope08] OpenLogic. Openlogic certification process. Whitepaper, October 2008.

[Ope11] OpenLogic. Open source software audits: Why, when and how to conduct an audit. Technical report, OpenLogic, 2011.

[P+99] Bruce Perens et al. The open source definition. *Open sources: voices from the open source revolution*, 1:171–85, 1999.

[Pal03] Doug Palmer. Why not use the gpl? thoughts on free and open-source software, 2003.

[Pat13] End Software Patents. Esp wiki. *ESP Wiki*, 1:1, 2013.

[Per05] Bruce Perens. The emerging economic paradigm of open source. *First Monday*, 10(10), 2005.

[Ray98] Eric S Raymond. Homesteading the noosphere. *First Monday*, 3(10-5), 1998.

[Ray99a] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999.

[Ray99b] Eric Raymond. The magic cauldron, 1999.

[RE04a] Lawrence Rosen and Michael B. Einschlag. Dealing with patents in software licenses. Internet, 4 2004.

[RE04b] C Ruffin and Christof Ebert. Using open source software in product development: A primer. *Software, IEEE*, 21(1):82–86, 2004.

[Rie07] Dirk Riehle. The economic motivation of open source software: Stakeholder perspectives. *Computer*, 40(4):25–32, 2007.

[Rie09] Dirk Riehle. The commercial open source business model. In *Value Creation in E-Business Management*, pages 18–30. Springer, 2009.

[Rie11] Dirk Riehle. Controlling and steering open source projects. *Computer*, 1:93–96, 2011.

[Rie12] Dirk Riehle. The single-vendor commercial open course business model. *Information Systems and e-Business Management*, 10(1):5–17, 2012.

[Ros04] Lawrence Rosen. *Open source licensing*. Prentice Hall PTR, 2004.

[SCA10] Alberto Simões, Nuno Carvalho, and José João Almeida. Testing as a certification approach. *ECEASST*, 33:10, 2010.

[Sch12] Mario Schaarschmidt. *Firms in Open Source Software Development*. Springer, 2012.

[Tri02] Leonard L. Tripp. Software certification debate: Benefits of certification. *IEEE Computer*, 35(6):31–33, 2002.

[V+05] Mikko Välimäki et al. *The rise of open source licensing: a challenge to the use of intellectual property in the software industry*. Helsinki University of Technology, 2005.

[Ver04] Martin Verwijmeren. Software component architecture in supply chain management. *Comput. Ind.*, 53(2):165–178, February 2004.

[Voa98] Jeffrey Voas. Cots software: The economical choice? *IEEE Softw.*, 15(2):16–19, March 1998.

[Voa00] Jeffrey Voas. Developing a usage-based software certification process. *Computer*, 33(8):32–37, 2000.

[Wal04] Kurt C Wallnau. Software component certification: 10 useful distinctions. *Software Engineering Institute*, 2004.

[WL01] Ming-Wei Wu and Ying-Dar Lin. Open source software development: an overview. *Computer*, 34(6):33–38, 2001.