FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Department Informatik

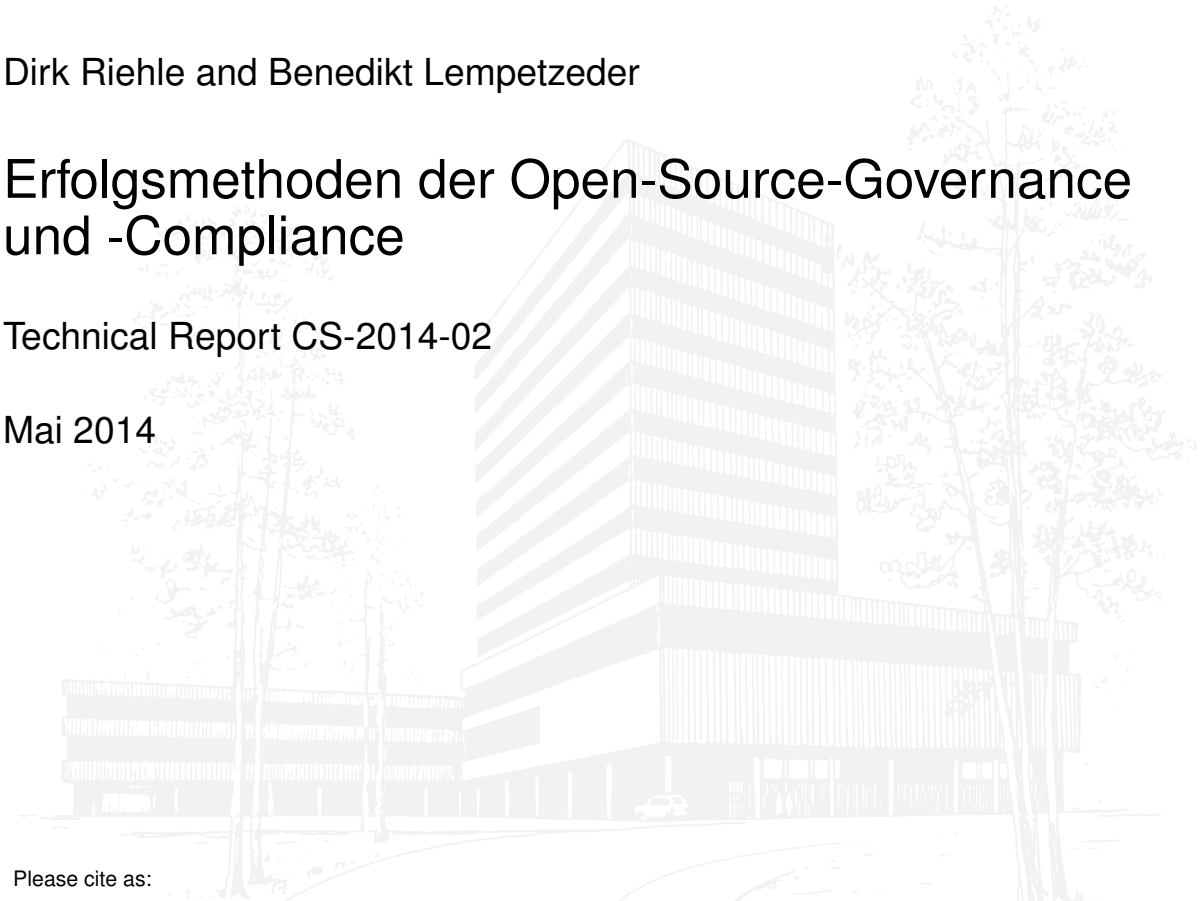
Technical Reports / ISSN 2191-5008

Dirk Riehle and Benedikt Lempetzeder

Erfolgsmethoden der Open-Source-Governance und -Compliance

Technical Report CS-2014-02

Mai 2014



Please cite as:
Dirk Riehle and Benedikt Lempetzeder, “Erfolgsmethoden der Open-Source-Governance und -Compliance,” Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Reports, CS-2014-02, Mai 2014.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik

Martensstr. 3 · 91058 Erlangen · Germany

www.cs.fau.de

ERFOLGSMETHODEN DER OPEN-SOURCE-GOVERNANCE UND -COMPLIANCE

Dirk Riehle
dirk@riehle.org, http://dirkriehle.com

Benedikt Lempetzeder
benedikt.lempetzeder@fau.de

Professorship for Open Source Software, Computer Science Department
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Zusammenfassung: Open-Source-Software ist weit verbreitet und wird nicht nur als alleinstehende Anwendungen eingesetzt, sondern auch als Komponenten in Produkten. Entsprechend wichtig ist es für Unternehmen, die Open-Source-Komponenten verwenden, mittels Open-Source-Governance und -Compliance sicherzustellen, dass die dem Einsatz von Open Source eigenen Risiken rechtzeitig und korrekt adressiert werden. Aufbauend auf früherer Arbeit zu den Risiken der Open-Source-Verwendung stellt dieser Artikel ein Modell und beispielhafte Erfolgsmethoden („Best Practices“) vor, mit denen Produktunternehmen diesen Risiken begegnen können. Das Modell und die Erfolgsmethoden stellen einen Auszug aus einem in Entwicklung befindlich Handbuch zur Open-Source-Governance und -Compliance dar. Weitere Information zu diesem Handbuch kann vom ersten Autor erfragt werden.

Stichworte: Open-Source-Governance, Open-Source-Compliance, Open-Source-Risiken, Open-Source-Chancen, Open-Source-Zertifizierung, Softwareökosystem, Softwarelieferanten, Softwarelieferkette, Open Source, Closed Source, Open-Source-Prozess, Open-Source-Praktiken, Erfolgsmethoden.

1. BEDEUTUNG VON OPEN-SOURCE-SOFTWARE

Open-Source-Software hat sich durchgesetzt; sie ist heutzutage überall zu finden:

- Bereits im Jahr 2010 nutzten 50% aller IT-Anwendungsunternehmen Open-Source-Software als normale Anwendungen neben Closed-Source-Software [Forrester 2009].
- IT-Anwendungsunternehmen schließen sich in Konsortien zusammen, um die Entwicklung von Open-Source-Software zu beauftragen [Heinritz et al. 2013].
- Bereits in 2012 setzten 80% aller Softwarehersteller Open-Source-Software in Ihren Produkten ein, um die Produkte schneller, besser, und kostengünstiger zu entwickeln [Gartner 2010].
- Softwarehersteller schließen sich in Konsortien zusammen, um die Entwicklung von Open-Source-Software zu koordinieren und durchzuführen.
- Softwarehersteller entwickeln neue Geschäftsmodelle, in denen die Open-Source-Software als Produkt der wesentliche Umsatzgenerator ist [Riehle 2012].

Open-Source-Softwareentwicklung ist öffentliche Arbeit im Internet. Somit sind diese Aussagen quantitativ-empirisch auf dem Internet messbar. Zwei Beispiele:

- Mind. 75% des neuen Quelltexts in einer der aktuellen Linux-Kernel-Releases wurde von Entwicklern eingereicht, die dafür von Unternehmen bezahlten wurden [Corbet et al. 2012].
- Eine konservative Abschätzung stellt fest, dass mind. 50% allen Open-Source-Quelltexts Mo-Fr von 9-17 Uhr geschrieben wird, also von Unternehmen bezahlt wird [Riehle et al. 2014].

Dieser Artikel diskutiert die Risiken, welche sich durch den Einsatz von Open-Source-Komponenten in Softwareprodukten ergeben und wie ihnen mit Methoden der Open-Source-Governance und -Compliance begegnet werden kann.

Im Rahmen der Open-Source-Governance legen Unternehmen ihre strategischen Ziele für den Umgang mit Open-Source-Software fest. Mit der Open-Source-Compliance und den zugehörigen Methoden versuchen sie, diese Ziele zu erreichen. Vereinfacht kann man strategische Ziele in die Bereiche Nutzung, Beitragen und Führen von Open-Source-Projekten einteilen. Die Nutzung und das taktische Beitragen zu Open-Source-Projekten fasst dieser Artikel unter „defensiver“ Open-Source-Governance zusammen. Das Initiieren und Führen von Open-Source-Projekten fällt unter die „proaktive“ oder „offensive“ Open-Source-Governance.

Korrekte defensive Open-Source-Governance und -Compliance gehört heutzutage zum Stand der Technik in der Produktentwicklung. Unternehmen, welche sie ignorieren, riskieren es, auf Schadenersatz verklagt zu werden. Proaktive, offensive Open-Source-Governance zählt noch als fortschrittlich und dem Stand der Technik voraus.

2. Open-Source-Governance und -Compliance

Unternehmen loben Open-Source-Software aufgrund der hohen Qualität und geringen Kosten. Aber Open-Source-Software birgt aus Softwareherstellersicht auch Gefahren.

2.1 Eigenschaften und Einsatz von Open-Source-Software

Open-Source-Software ist Software, die ihren Nutzern unter einer Open-Source-Lizenz zur Verfügung gestellt wird. Wesentliche Eigenschaften einer Open-Source-Lizenz sind:

- Die Software steht im Quelltext kostenfrei zur Verfügung.
- Die Software kann kostenfrei modifiziert oder unmodifiziert eingesetzt werden.
- Die Software kann kostenfrei mit oder ohne Modifizierung weitergegeben werden.
- Die Software kann in beliebigen Anwendungsfeldern eingesetzt werden.

Die meisten Nutzer freuen sich darüber, dass ihnen keine, kaum, oder im Vergleich zu Closed-Source-Software verringerte Kosten entstehen. Obendrein ist die Qualität von Open-Source-Software häufig höher als die von vergleichbarer Closed-Source-Software, da die Open-Source-Software öffentlich entwickelt und bereitgestellt wird. Dies hat zur Folge, dass eine potentiell große Anzahl von Nutzern Fehler schnell findet. Da sich Entwickler nicht gern öffentlich blamieren, sind die Quelltextbeiträge häufig von höherer Qualität als innerhalb von Unternehmen, wo man sich ggf. nur im kleinen Kreis bloßstellt.

Neben Rechten legen Open-Source-Lizenzen den Nutzern der Software auch Pflichten auf. So verlangen z.B. fast alle Open-Source-Lizenzen die sog. „Attributierung“, also bei Verwendung der Open-Source-Software in Produkten die angemessene Bereitstellung der Information, welche Open-Source-Komponenten unter welcher Lizenz verwendet wurden. Aus diesen Gründen sieht man heutzutage bei kommerzieller Closed-Source-Software immer auch eine Liste von Open-Source-Komponenten, die in den Produkten verwendet wurden, inklusive der dazugehörigen Lizenzen. Diese Listen werden immer länger, wie das Beispiel der von der Daimler AG verwendeten Open-Source-Software[1] zeigt.

Setzt ein Softwarehersteller eine Open-Source-Komponente in seinen Produkten ein, dann gehört das Ausführen der Attributierung zur guten Open-Source-Governance und -Compliance. Ein Verstoß gegen die Attributierungspflicht könnte ggf. an- und kostenpflichtig abgemahnt werden.

Eine weitaus schwerwiegendere Verpflichtung ergibt sich über das sog. „Copyleft“, auch „virale“ Lizenzklausel genannt. Hierbei handelt es sich um eine Klausel, welche den Nutzer von Open-Source-Software, z.B. einen Produkthersteller, bei Weitergabe der Open-Source-Software verpflichtet, etwaigen assoziierten proprietären Quelltext ebenfalls unter einer Open-Source-Lizenz bereitzustellen (der erwähnte „virale“ Effekt).

Eine Open-Source-Komponente gilt z.B. als „weitergegeben“, wenn sie in einem kommerziellen Produkt eingebaut und verkauft wird. In Abhängigkeit von der technischen Kopplung des proprietären

1 Siehe http://www4.mercedes-benz.com/manual-cars/ba/foss/content/en/licence_agreement.html

Quelltexts mit dem Open-Source-Quelltext kann es jetzt passieren, dass der proprietäre Quelltext, also das geistige Eigentum des Herstellers, ebenfalls kostenfrei unter einer Open-Source-Lizenz bereitgestellt werden muss. Da dieses geistige Eigentum häufig wettbewerbsdifferenzierend ist, stellt sich diese Konsequenz zumeist als inakzeptabel dar.

2.2 Risiken und Herausforderungen für Hersteller

In früherer Arbeit haben wir diese Risiken und ihre Konsequenzen herausgearbeitet [Helmreich und Riehle 2012]. Abbildung 1 stellt diese Risiken in einer Abhängigkeitskette dar:

Abbildung 1: Risiken, welche Open-Source-Software für Produkthersteller darstellt (aus [Helmreich und Riehle 2012])

Eine umfangreiche Beschreibung würde den verfügbaren Rahmen sprengen, weswegen wir nur jeweils ein Beispiel anführen:

1. ***Unkontrollierte Verwendung („non-governed use“).*** Fügt ein Entwickler unkontrolliert Open-Source-Quelltext in den Produkt-Quelltext ein, fehlt jegliche Nachvollziehbarkeit. Werden später Fehler und Sicherheitslücken im Open-Source-Quelltext bekannt, können diese nicht oder nur mit Mühe im Produkt-Quelltext aufgespürt werden.
2. ***Unkontrollierte Beiträge („non-governed contributions“).*** Beteiligt sich ein Entwickler auf Open-Source-Mailing-Listen aktiv oder trägt gar unkontrolliert Quelltext zu Open-Source-Projekten bei, riskiert er oder sie, geistiges Eigentum des Unternehmens offenzulegen resp. zu verschenken. Konkurrenten können daran z.B. frühzeitig Geschäftsstrategien erkennen.
3. ***Gegenstand einer Klage („subjected to lawsuit“).*** Kommt es zu einer Klage wegen (vermuteten) Verstoßes gegen eine oder mehrere Open-Source-Lizenzen findet sich das Unternehmen ggf. im Gerichtssaal wieder. Dies verursacht Kosten und verlangt Aufwand vom Management, was alles dem Unternehmenserfolg abträglich ist.
4. ***Verurteilung, Quelltext offenzulegen („having to open source“).*** Verliert der Hersteller das Gerichtsverfahren, wird er ggf. gezwungen, proprietären Quelltext unter einer Open-Source-Lizenz offenzulegen. Wiederum können Konkurrenten Strategien daran erkennen, Patentverletzungen ableiten oder sich gar den Wert der Arbeit aneignen.
5. ***Verurteilung wegen Patentverletzung („convicted of infringing patents“).*** Ist eine der Konsequenzen eines verlorenen Gerichtsverfahrens, den Quelltext offenlegen zu müssen, werden allfällige Patentverletzungen offenkundig. Entsprechend stehen weitere Gerichtsverfahren ins Haus oder auch gleich Patentlizenzkosten, denen man sich nicht entziehen kann.

Eine aus dem Jahr 2012 stammende Studie von BearingPoint, welche wir begleitet haben, zeigt auf, dass die europäische Automobilbranche zwar stark auf Open-Source-Software setzt, aber gleichzeitig noch großen Entwicklungsbedarf bzgl. Open-Source-Governance und -Compliance hat [BearingPoint 2012].

Trotz dieser Risiken ist die Bedeutung von Open Source weltweit signifikant gewachsen und wird auch weiter wachsen. Dies ist nur möglich, weil Hersteller Mittel und Wege gefunden haben, mittels Erfolgsmethoden der Open-Source-Governance und -Compliance diesen Risiken konstruktiv und effektiv zu begegnen.

3. Ein Modell für die Open-Source-Governance und -Compliance

Open-Source-Governance umfasst die strategischen Ziele, die ein Unternehmen in der Nutzung von, dem Beitragen zu, und dem Führen von Open-Source-Projekten sieht. Open-Source-Compliance umfasst die Methoden, welche zur Zielerfüllung führen sollen. Da die meisten Unternehmen rechtskonform sein wollen, umfasst die Open-Source-Governance auch das Nicht-Verstoßen gegen Lizenzen als Ziel und somit die Compliance das Einhalten der Pflichten dieser Lizenzen resp. das Vermeiden von Situationen, in denen ungewollte Pflichten erfüllt werden müssen, um rechtskonform zu sein.

3.1 Rollen und Zuständigkeiten im Unternehmen

Jedes Produktunternehmen verfügt über die Rolle des oder der Entwicklerin. Für die Open-Source-Governance und -Compliance werden weitere Rollen benötigt. Spezifisch sind dies:

- Eine oder einen Open-Source-Kontrolleur (Compliance-Verantwortlichen) sowie
- Eine oder einen im Open Source kompetenten Rechtsanwalt.

Die Kontrolleur-Rolle sollte nicht normalen Entwicklern oder Entwicklungsleitern zugewiesen werden, sondern zur Vermeidung von Interessenskonflikten als Stabsfunktion aus der normalen Hierarchie herausgehoben werden.

Die Rechtsanwalt- und Kontrolleur-Rollen können im Prinzip von der selben Person gespielt werden. Dies geschieht allerdings selten, da die notwendigen Kompetenzen (tiefes technisches und rechtliches Verständnis) fast nie in einer Person zu finden sind.

Es ist weiterhin nicht ratsam, Open-Source-Compliance vollständig in der Rechtsabteilung zu verankern, da diese üblicherweise stark auf Risikoreduzierung fokussiert ist und Geschäftschancen oft nicht erkennen oder bewerten kann.

Die Definition der Open-Source-Governance sollte durch den CTO unter Einbeziehung der Geschäftsführung und in Zusammenarbeit mit Entwicklungsleitern und besagtem Open-Source-Verantwortlichen und -Rechtsanwalt geschehen. Die Umsetzung obliegt dann dem Open-Source-Verantwortlichen, dem auch entsprechende Rechte zugestanden werden müssen.

In großen Entwicklungsorganisationen werden häufig zusätzlich zu der zentralen Kontrolleur-Rolle dezentrale Rollen etabliert, die das Tagesgeschäft abwickeln und sich mit der zentralen Rolle zum Wissens- und Erfahrungsaustausch vernetzen.

3.2 Softwarelieferanten und -Abnehmer

Probleme durch Open Source entstehen primär, wenn ungewollter Open-Source-Quelltext im Produktquelltext landet. Hier gibt es mehrere Wege von Open-Source-Quelltext in ein Produkt:

1. ***Händisches Kopieren.*** Ein Entwickler kopiert ungewollten Open-Source-Quelltext in Produktquelltext, für den er oder sie zuständig ist.
2. ***Einbettung einer Open-Source-Komponente.*** Ein Entwickler pflegt eine Open-Source-Komponente in das Produkt ein.
3. ***Einbettung über Drittparteien.*** Eine Open-Source-Komponente wird als Teil von Fremdsoftware eingepflegt.

Open-Source-Projekte sollten wie Closed-Source-Software von Drittparteien als Lieferanten gesehen werden. Das Produktunternehmen wird zum Abnehmer und muss einen entsprechenden Abnahmeprozess definieren. Misstrauen gegenüber deklarierten Inhalten ist sowohl bei Open-Source- wie bei Closed-Source-Komponenten angebracht.

Während ein Unternehmen durch Ausbildung die eigenen Entwickler für den korrekten Umgang mit Open-Source-Software sensibilisieren kann, ist dies für Fremdlieferanten häufig nicht möglich.

1. Definiere strategische und technische Ziele der Open-Source-Nutzung
2. Definiere Rollen und Zuständigkeiten für Prozesse der Open-Source-Nutzung
3. Definiere Anforderungen an Drittparteiencode und -komponenten
4. Definiere Prüfprozess für Komponenten
    a. Definiere Anlaufstelle in Rechtsabteilung
    b. Gib Rechtsabteilung Veto-Recht für den Einsatz von Drittparteienkomponenten
5. Etabliere klare vertragliche Beziehung mit kommerziellen Lieferanten
    a. Sichere Anforderungen an kommerzielle Drittparteienkomponenten vertraglich ab
    b. Sichere Bereitstellung von Stückliste bei Lieferung vertraglich ab
    c. Definiere Vertragsstrafen bei Verstoß gegen Anforderungen
    d. Verpflichte Lieferanten auf Einhaltung von Erfolgsmethoden der Open-Source-Compliance
    e. Sichere Überraschungsaudits für Open-Source-Compliance ab
6. Kontrolliere den Zufluss von Drittparteiencode
    a. Wende Prüfprozess auf jede offiziell einlaufende Drittparteienkomponente an
        i. Analysiere Open-Source-Komponenten auf Anforderungen hin
        ii. Überprüfe Closed-Source-Komponenten anhand der Stückliste
        iii. Überprüfe alle Softwarekomponenten mittels Quelltext-Scan, sofern möglich
    b. Überprüfe alle internen Commits mittels Quelltext-Scanner
7. Verwalte Open-Source-Komponenten
    a. Verwalte alle Metadaten (Code-Label) für akzeptierte Komponenten
    b. Stelle akzeptierte Komponenten an bekanntem Ort bereit
    c. Stelle Liste nicht-akzeptierter Komponenten an bekanntem Ort bereit
8. Überprüfe kommerzielle Lieferanten in unregelmäßigen Abständen durch Überraschungsaudits
9. Bilde alle Stakeholder des Entwicklungsprozesses aus
    a. Kommuniziere Ziele der Open-Source-Nutzung
    b. Kommuniziere Risiken von Drittparteienkomponenten
    c. Kommuniziere Prüfprozesse
    d. Bilde neue Entwickler bei Eintritt aus
    e. Wiederhole Ausbildung in regelmäßigen Abständen
10. Überwache Open-Source-Projekte hinsichtlich Bekanntwerden von Sicherheitslücken etc.

Tabelle 1: Ausgewählte defensive Erfolgsmethoden der Open-Source-Compliance

Das Paar (Lieferant, Abnehmer) setzt sich rekursiv in einer Lieferkette fort. Es gilt sowohl innerhalb (großer) Unternehmen wie auch über Unternehmensgrenzen hinweg; entsprechend muss die Lieferkette ganzheitlich betrachtet werden.

Häufig gibt es in Softwarelieferketten einen exponierten Abnehmer in der Kette, welcher mit viel Marktmacht in diese Lieferkette eingreifen kann. Compliance-Audits sind eine, wenngleich teure, Problemlösung für diesen Abnehmer. An anderer Stelle unterbreiten wir deswegen einen Alternativvorschlag mittels Open-Source-Zertifizierung der Softwarelieferkette [Lempetzeder 2013].

Im Folgenden adressieren wir diese und andere Herausforderungen.

1. Definiere Ziele der Open-Source-Nutzung	
Kontext	Ein Produktersteller, der sich in aktivem Wettbewerb sieht und für den die Qualität seiner Software ein wesentlicher Wettbewerbsdifferenzierer ist. [...]
Problem	Immer komplexer werdende Software verlangt zunehmende Investitionen in Bereichen, die nicht wettbewerbsdifferenzierend für die Produkte des Unternehmens sind. [...]
Lösung	Setze Open-Source-Software in den nicht wettbewerbsdifferenzierenden Bereichen ein. Trage die Entwicklung der Software mit. Definiere und kommuniziere die Ziele des Einsatzes. [...]

5. Etabliere klare vertragliche Beziehung mit kommerziellen Lieferanten	
Kontext	Ein Unternehmen, welches in seinen Produkten Software von Drittparteien einsetzt, über deren Entwicklungsprozess es keine direkte Kontrolle hat. [...]
Problem	Die Entwicklung der Drittsoftware folgt nicht den eigenen Compliance-Anforderungen. Somit stehen ggf. ungewünschte Überraschungen bei Abnahme ins Haus. [...]
Lösung	Regele die Verwendung von Open-Source-Komponenten in der Drittsoftware. Definiere, was erlaubt ist [...] Verlange Stückliste bei Lieferung. Sichere Audits vertraglich ab. [...]

5b. Sichere Bereitstellung von Stückliste bei Lieferung vertraglich ab	
Kontext	Ein Unternehmen, welches die Lieferung von Software Drittparteien vertraglich abgesichert hat. [...]
Problem	Der Abnahmeprozess muss definiert sein und Vertragsverstöße müssen klar erkennbar sein. [...]
Lösung	Verlange eine korrekte Stückliste unter Verwendung von SPDX. [...]

7. Verwalte Open-Source-Komponenten	
Kontext	Ein Unternehmen, welches Open-Source-Komponenten in seinen Produkten einsetzen möchte. [...]
Problem	Viele Komponenten werden wiederholt eingesetzt und sollten nur einmal überprüft werden. [...]
Lösung	Führe eine entsprechende Liste akzeptierter und nicht-akzeptierter Komponenten mit dokumentiertem Entscheidungsprozess. [...]

7b. Stelle akzeptierte Komponenten an bekanntem Ort bereit	
Kontext	Ein Unternehmen, welches Open-Source-Komponenten in seinen Produkten einsetzen möchte. [...]
Problem	Entwickler vermeiden die Formalia, welche vor dem Einsatz von Open-Source-Komponenten stehen. [...]
Lösung	Stelle einmal akzeptierte Komponenten in einem Repository an bekannter Stelle für alle sichtbar zur Verfügung. [...]

Tabelle 2: Skizzen ausgewählter Erfolgsmethoden der defensiven Open-Source-Compliance

ID:	5.b.i
Name:	Verwendung eines Standards für Stücklisten für Drittparteienkomponenten
Zusammenfassung:	Externe Komponenten sollten mit definierter Stückliste („bill of materials“) versehen kommen und diese Stückliste sollte einem einheitlichen Standard genügen, um die Weiterverarbeitung der Information zu vereinfachen.
Voraussetzungen:	3.e. Sichere Verfügbarkeit validierter Stückliste für verwendete Open-Source-Komponenten ab 5.b. Sichere Bereitstellung von Stückliste bei Lieferungen kommerzieller Lieferanten vertraglich ab [...]
Kontext:	Einlaufende Komponenten müssen auf korrekten Inhalt und Kompatibilität mit Open-Source-Strategie, Unternehmensstandards und Projektanforderungen hin überprüft werden.
Problem:	Schnelle Releasezyklen im Open Source und ggf. fehlende Verfügbarkeit des Quelltexts im Closed Source verlangen einen Prüfprozess, der effizient und effektiv umgesetzt werden kann.
Lösung:	In Vorbereitung eines effizient Prüfprozesses, greife auf etablierte resp. sich etablierende Standards zur Stücklistenbeschreibung zurück. Achte darauf, das aktuelle oder zukünftige Werkzeuge das zum Standard gehörige Datenformat lesen und weiterverarbeiten können. [...]
Konsequenzen:	6.a. Wende Prüfprozess auf alle einlaufenden Komponenten an 7.h. Definiere Werkzeugkette für Open-Source-Prozesse [...]
Beispiele:	SPDX, die „Software Package Data Exchange“ Spezifikation (siehe http://spdx.org) ist ein von mehreren Herstellern getragener sich etablierender Standard zur Definition von Stücklisten. Samsung berichtet über seine Erfahrungen mit dem Einsatz von SPDX im Rahmen ihrer Open-Source-Prozesse (siehe „Piloting SPDX in Samsung: Case Studies and Experiences“). [...]

Tabelle 3: Auszug aus einer ausgearbeiteten Erfolgsmethode in Musterform

4. Erfolgsmethoden der Governance und -Compliance

Zur Unterstützung von Produktunternehmen beabsichtigen wir, ein Handbuch mit Erfolgsmethoden in der Open-Source-Governance und -Compliance zu entwickeln. In diesem Abschnitt adressieren wir den Bereich der „defensiven“ Open-Source-Nutzung.

4.1 Erfolgsmethoden Defensiver Open-Source-Governance

Tabelle 1 stellt einen Auszug aus dem erwähnten Handbuch für die Erfolgsmethoden vor. Was sich zuerst als lineare Liste darstellt, wird in der endgültigen Form des Handbuchs ein Graph voneinander abhängiger Erfolgsmethoden werden. Die Abhängigkeiten im Graphen beschreiben, welche Erfolgsmethoden welche andere bedingen oder ihnen vorangehen. Abbildung 2 skizziert dies.

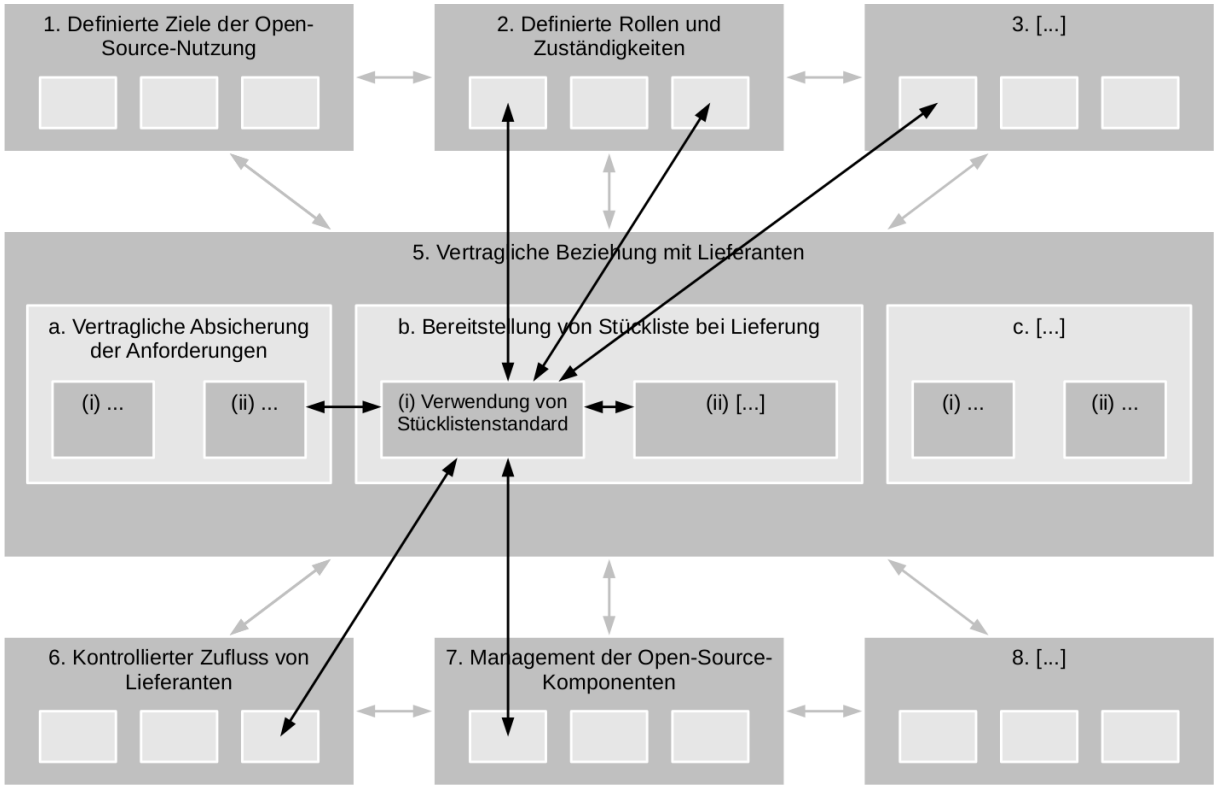


Abbildung 2: Skizze der Beziehungen einer Erfolgsmethode im Kontext eines Handbuchs

4.2 Ausgewählte Beispiele der Erfolgsmethoden

Tabelle 2 stellt einige ausgewählte Erfolgsmethoden als Skizze zur Verfügung. Die Erfolgsmethoden werden nach dem aus der Softwaremustergemeinde bekannten Prinzip des Kontext-Problem-Lösung-Tripels dargestellt.

Tabelle 3 stellt eine einzelne Erfolgsmethode in etwas mehr Detail vor. Auch diese Erfolgsmethode muss natürlich noch weiter ausgearbeitet werden. Diese Ausarbeitung stellt einen Großteil der Arbeit des erwähnten Handbuchs dar.

5. Zertifizierung und Offensiver Einsatz

Jenseits der in diesem Artikel diskutierten defensiven Open-Source-Nutzung möchten wir auf zwei relevante zukünftige Aufgaben hinweisen:

- ***Zertifizierung von Open-Source-Governance und -Compliance.*** Wie illustriert, ist die Überwachung der Lieferkette mittels Audits Stand der Technik. Leider sind Audits teuer. Aus Abnehmersicht wäre es besser, wenn die beteiligten Unternehmen sich bzgl. guter Open-Source-Governance und -Compliance zertifizieren lassen würden. Wir haben dies bereits vorgedacht, siehe [Helmreich 2011] und [Lempetzeder 2013]; es muss aber noch viel getan werden.
- ***Strategischer offensiver Einsatz von Open-Source-Projekten.*** Unternehmen wie IBM und Rackspace machen es vor: Open Source ist eine probates Mittel, um Industrieplattformen zu etablieren, sich Marktmacht anzueignen und entsprechenden Umsatz zu erwirtschaften. Dies ist für jedes Unternehmen möglich, ob groß oder klein. Wir beabsichtigen ein entsprechendes Handbuch auf für diesen Einsatz von Open Source zu entwickeln.

Open-Source-Governance und -Compliance ist ein reichhaltiges Betätigungsfeld; wir befinden uns vielfach noch am Anfang. Es stehen allen Beteiligten spannende Zeiten ins Haus!

6. DANKSAGUNG

Wir bedanken uns bei Martin Helmreich, Michael Philippsen sowie den anonymen Gutachtern zur Ausgabe 297 von HMD für das hilfreiche Feedback, welches uns half, diesen Artikel zu verbessern und in seine endgültige Fassung zu bringen.

LITERATUR

[BearingPoint 2012] BearingPoint. *FOSS Management Study*. BearingPoint GmbH, 2012. Available from http://www.bearingpoint.com/en-other/7-5601/study-foss-management/

[Corbet et al. 2012] J. Corbet, G. Kroah-Hartman, and A. McPherson, „Linux Kernel Development – How fast it is going, who is doing it, what they are doing, and who is sponsoring it?“ Linux Foundation, 2012. Available from http://go.linuxfoundation.org/who-writes-linux-2012

[Forrester 2009] Jeffrey S. Hammond. „Open Source Software Goes Mainstream.“ Forrester Research, 2009.

[Gartner 2010] Mark Driver. „Key Issues for Open Source Software, 2010.“ Gartner Research: 2010.

[Heinritz et al. 2013] Heinritz, Christof; Herdt, Peter; Janeck, Stephan; Regenbogen, Gerhardt; Riehle, Dirk; Rose, Frank; Roth, Michael; Thoma, Detlef; Tuchs, Michael (2013). Konsortiale Open-Source-Softwareentwicklung im Energiesektor. Available from http://www.osbf.eu/blog/konsortiale-open-source-softwareentwicklung/im-energiesektor/

[Helmreich 2011] Helmreich, Martin. Best Practices of Adopting Open Source Software in Closed Source Software Products. Diplomarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg: 2011.

[Helmreich und Riehle 2012] Helmreich, Martin; Riehle, Dirk (2012). Geschäftsrisiken und Governance von Open-Source in Softwareprodukten. HMD 283, 17-25. Available from http://dirkriehle.com/2011/12/17/business-risks-and-governance-of-open-source-in-software-products-in-german/

[Lempetzeder 2013] Lempetzeder, Benedikt. Zertifizierung von Open Source Lieferketten. Master Thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg: 2013.

[Riehle 2012] Riehle, Dirk (2012). The Single-Vendor Commercial Open Source Business Model. Information Systems and e-Business Management vol. 10, no. 1. Springer Verlag, 2012, 5-17. Available from http://dirkriehle.com/2010/12/06/the-single-vendor-commercial-open-source-business-model/

[Riehle et al. 2014] Riehle, Dirk; Riemer, Philipp; Kolassa, Carsten; Schmidt, Michael (2014). Paid vs. Volunteer Work in Open Source. In Proceedings of the 47th Hawaii International Conference on System Science (HICSS 2014). IEEE Press, 2014. Available from http://dirkriehle.com/2013/08/22/paid-vs-volunteer-work-in-open-source/