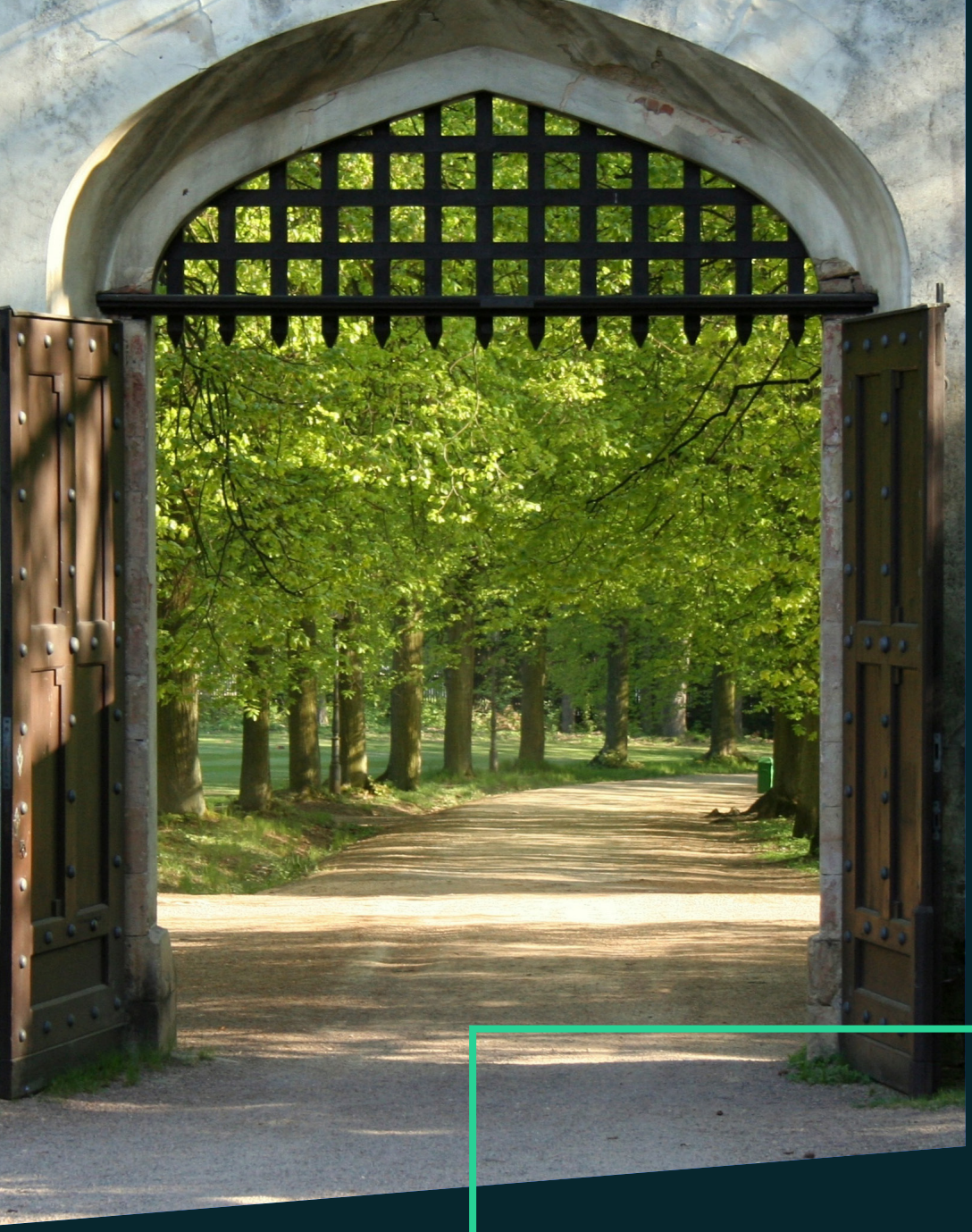

bitkom

Open-Source-Leitfaden

Praxisempfehlungen für
Open-Source-Software
Version 3.2

Am Anfang war alle Software frei.

Georg C. F. Greve

Herausgeber
Bitkom e. V.
Albrechtstraße 10 | 10117 Berlin

Ansprechpartner
Dr. Frank Termer | Bereichsleiter Software
T 030 27576-232 | f.termer@bitkom.org

Verantwortliches Bitkom-Gremium
AK Open Source

Projektleitung
Sebastian Hetze | Red Hat GmbH

Gestaltung
Daniel Vandré

Titelbild
© kkolis – stock.adobe.com

Geleitwort

Der vorliegende Leitfaden zu Open-Source-Software (OSS) wurde im Arbeitskreis Open Source erarbeitet. In diesem Arbeitskreis versammeln sich Expertinnen und Experten der Bitkom-Mitgliedsunternehmen, die sich in ihrer beruflichen Tätigkeit intensiv mit dem Einsatz von OSS in einem professionellen Unternehmensumfeld auseinander setzen. Die Erstellung des Leitfadens beruht jedoch zu einem überwiegenden Teil auf ihrem persönlichen ehrenamtlichen Engagement. In den zurückliegenden Monaten, teilweise Jahren (erste Idee zur Überarbeitung des bestehenden Leitfadens wurden bereits im Februar 2020 gesammelt und ausgetauscht), haben sich alle Beteiligten mit sehr hohem Einsatz, großer Liebe zum Detail und mit höchster Professionalität dem Projekt der Überarbeitung des Leitfadens gewidmet.

Dabei wurde mit viel Leidenschaft um die beste Struktur, die besten Inhalten und die besten Formulierungen gerungen und in unzähligen Telefon- und Videokonferenzen, in Issues und Pull Request auf GitHub sowie in Mails und Nachrichten Standpunkte, Sichtweisen und Meinungen ausgetauscht und konsensorientiert Lösungen gefunden. Das Ergebnis dieses nicht immer ganz einfachen Prozesses können Sie nun in den Händen halten!

Wir hoffen, mit diesem Leitfaden dem Themenfeld Open-Source-Software gleichzeitig allgemeinverständlich, aber trotzdem mit hohem fachlichen Anspruch gerecht zu werden. Es ist explizites Anliegen, das Dokument künftig laufend fortzuschreiben und zu aktualisieren, um es stets auf einem aktuellen Stand zu halten. Wir laden daher alle Interessierten herzlich ein, sich an der Weiterentwicklung des Leitfadens zu beteiligen. Für Fragen, aber auch für kritische Anmerkungen zu den Inhalten, stehen wir gern jederzeit zur Verfügung.

- Dr.-Ing. Marius Brehler, Fraunhofer-Institut für Materialfluss und Logistik IML
- Prof. Dr. Christian Czychowski, Nordemann Czychowski & Partner Rechtsanwältinnen und Rechtsanwälte Partnerschaft mbB
- Sebastian Dworschak, Nordemann Czychowski & Partner Rechtsanwältinnen und Rechtsanwälte Partnerschaft mbB
- Oliver Fendt, Siemens AG
- Dr. Lars Geyer-Blaumeiser, Robert Bosch GmbH
- Sebastian Hetze, Red Hat GmbH
- Holger Koch, DB Systel GmbH
- Cédric Ludwig, Osborne Clarke
- Dr. Marc Ohm, Fraunhofer-Institut für Kommunikation, Informationsverarbeitung und Ergonomie FKIE & Universität Bonn
- Michael Picht, SAP SE
- Karsten Reincke, Deutsche Telekom AG
- Julian Schauder, PricewaterhouseCoopers GmbH Wirtschaftsprüfungsgesellschaft
- Marcel Scholze, PricewaterhouseCoopers GmbH Wirtschaftsprüfungsgesellschaft
- Thomas Schulte, metaeffekt GmbH
- Cornelius Schumacher, DB Systel GmbH
- Dr. Hendrik Schöttle, Osborne Clarke
- Christof Walter, SAP SE

Danksagung

Der Bitkom-Leitfaden zu Open-Source-Software in der Version 3.0 setzt durchaus auf der Vorversion auf. So danken wir an dieser Stelle auch den Autorinnen und Autoren der Version 2.0 aus dem Jahr 2016.

Dr. Oliver Block, Bundesdruckerei GmbH | Antonia Byrne, Capgemini Deutschland Holding GmbH | Oliver Fendt, Siemens AG | Christine Forster, DATEV eG | Sigrid Freidinger, Nokia Solutions and Networks GmbH & Co. KG | Dr. Martin Greßlin, SKW Schwarz Rechtsanwälte | Björn Hajek, LL.M. (University College London), Infineon Technologies AG | Dr. Michael Jäger, Siemens AG | Sylvia F. Jakob LL.M. (Edinburgh), Institute for Legal Informatics, Leibniz Universität Hannover | Katharina Komarnicki, Siemens AG | Claudia Krell, SerNet GmbH | Marco Lechner, Accenture GmbH | Dr. Johannes Loxen, SerNet GmbH | Felix Mannewitz, Siemens AG | Karsten Reincke, Deutsche Telekom AG | Monika Schnizer, Fujitsu Technology Solutions GmbH | Dr. Hendrik Schöttle, Osborne Clarke | Sonja Schlerkmann, LL.M., Vodafone Kabel Deutschland GmbH | Martin Schweinoch, SKW Schwarz Rechtsanwälte | Udo Steger, Unify GmbH & Co. KG | Maximilian Stiegler, Océ Printing Systems GmbH & Co. KG | Axel Teichert, DB Systel GmbH | Dr. Christian-David Wagner, Wagner Rechtsanwälte | Dr. Hans Peter Wiesemann, DLA Piper UK LLP

Berlin, im Juni 2022

Changelog

Alle bemerkenswerten Änderungen am Open-Source-Leitfaden werden auf dieser Seite dokumentiert. Ihr Format wurde von ↗ Führe ein Changelog übernommen. Die Versionsnummern dieses Leitfadens werden entsprechend ↗ Semantic Versioning vergeben.

[3.0] – 30.09.2022

Geändert

- Der gesamte Leitfaden wurde gegenüber früheren Versionen grundlegend überarbeitet.

[3.1] – 28.02.2023

Geändert

- Das Kapitel »Software Bill of Materials (SBOM)« wurde als Kapitel 9.2 eingefügt.
- Das Kapitel »Exkurs: Zur Entstehung von Open-Source-Software« wurde auf die Ebene 9.1 verschoben.
- Es wurde eine Seite »Changelog« eingefügt.
- Das Stichwortverzeichnis wurde aktualisiert.

[3.2] – 28.08.2024

Geändert

- Das Glossar wurde im Anhang ergänzt.

1 Einleitung | Vorwort

Dieser Leitfaden stammt im Original aus dem Jahr 2016 und sechs Jahre später wurde es höchste Zeit, ihn zu überarbeiten. Denn mittlerweile haben sich die Entwicklungen rund um »Open Source« massiv beschleunigt und es ist auf breiter Front die Erkenntnis gereift, dass eine digitale Innovation ohne Open-Source-Software kosteneffizient nicht möglich ist.

Deutlich aufgezeigt wurde das auch durch den *Bitkom Open-Source-Monitor*, der 2021 bereits zum zweiten Mal durchgeführten Studie zu Open-Source-Software in Deutschland.[1] Open-Source-Software wird – branchenübergreifend – in nahezu jedem Entwicklungsprojekt eingesetzt und ist Bestandteil fast aller Softwareprodukte. Auch die aktive Beteiligung an Open-Source-Projekten und der strategische Einsatz von Open-Source-Methodik ist heute weit verbreitet. Anders gesagt: Open Source hat auf ganzer Linie gewonnen.

Um dieser Realität gerecht zu werden, legt der Bitkom Arbeitskreis Open Source eine umfangreich überarbeitete Neufassung seines Leitfadens vor.

Dieser Leitfaden gibt einen breiten Überblick über die vielerorts etablierte Praxis der Open-Source-Nutzung und deren rechtliche Implikationen[2]. Er beleuchtet Erfahrungen mit Open-Source-Software, verweist auf *Best-Practices* für die aktive Beteiligung an Open-Source-Entwicklungen und skizziert Möglichkeiten, Open-Source-Projekte zu steuern. Außerdem zeigt er Möglichkeiten zur strategischen Verwendung von Open-Source-Software und -Methodiken auf, und zwar bis hinein in neue Geschäftsmodelle.

Open-Source-Software erfolgreich einzusetzen, heißt letztlich, das kooperative Open-Source-Spiel der freien Nutzung, der freien Weitergabe und Veränderung mitzuspielen. Wer es mitspielt, macht die Erfahrung, dass der eigene Nutzen die Kosten weit übersteigt, die sich aus den Lizenz bedingten Forderungen ergeben. Die Teilnehmer und Teilnehmerinnen dieses ›Spiels‹ werden die Erfahrung machen, dass Sie mehr aus der Community bekommen, als das, was sie investieren. So können alle diese Rechnung aufmachen und ihren persönlichen positiven Saldo feststellen.

In diesem breit gefächerten Sinne richtet sich der Leitfaden an

- Entscheiderinnen und Entscheider,
- Projektleitende sowie
- Entwicklerinnen und Entwickler

aus Wirtschaft und Verwaltung, die Open-Source-Software und Open-Source-Methodiken nutzen und deren Vorteile erschließen wollen.

So wünscht Ihnen der Vorstand vom AK Open Source viel Freude und gute Erkenntnisse, wenn Sie unseren Bitkom-Leitfaden Open-Source-Software in der Version 3.0 zurate ziehen. Gern steht Ihnen der gesamte Vorstand für Rückfragen und Anmerkungen zur Verfügung, und wir laden Sie herzlich zur aktiven Mitarbeit im Arbeitskreis Open Source ein.[3]

1 https://www.bitkom.org/openSourcemonitor
2 Für eine ergänzende Betrachtung der rechtlichen Aspekte sei auf den Kurzleitfaden ↗ »Open-Source-Software – Rechtliche Grundlagen und Handlungshinweise« verwiesen
3 https://www.bitkom.org/Bitkom/Organisation/Gremien/Open-Source.html

2 Info & Grundlagen

2.1 Relevanz von Open-Source-Software

Im digitalen Zeitalter ist die Relevanz von Software universell. In den hoch industrialisierten Staaten gibt es kaum noch Lebens-, Arbeits- oder Freizeitbereiche, in denen digitale Prozesse, Assistenten oder Widgets nicht etabliert sind. Das, was wir heute als ›Digitalisierung‹ bezeichnen, wird zu einem großen Teil von **Open-Source-Software** betrieben und getrieben: Milliarden von Menschen haben Open-Source-Software in ihren Taschen in Form eines Smartphone-Betriebssystems. Die Cloud läuft zum größten Teil auf Open-Source-Software[4]. Selbst auf dem Mars ist sie als Teil des Mars-Helikopters Ingenuity[5] inzwischen angekommen.

Studien zeigen, dass in fast allen Unternehmen direkt oder indirekt Open-Source-Software zum Einsatz kommt. So ergab etwa der Bitkom Open-Source-Monitor 2021[6], dass 87 % aller großen deutschen Unternehmen bewusst Open-Source-Software einsetzen.

87 %

aller großen deutschen Unternehmen setzen bewusst Open-Source-Software ein.

Open-Source-Software steht proprietärer Software in puncto **Zuverlässigkeit und Sicherheit** in nichts nach. In vielen Bereichen zeigt sie sich als **Innovationsmotor** mit ungeheurem Potenzial. Neben der **Vielfalt der Anwendungsszenarien** steht die **Fülle der Anwendungen**. Open-Source-Software hat **echte wirtschaftliche Bedeutung** gewonnen.

Die Gründe dafür sind simpel:

- Die Software ist frei nutzbar, es fallen weder Lizenzkosten an, noch besteht das Risiko, dass die Nutzung der aktuellen Version durch den Urheber in Zukunft eingeschränkt wird.
- Durch die Verwendung standardisierter Open-Source-Lizenzen sind keine individuellen Vertragsverhandlungen nötig.
- Offene Entwicklungsmodelle geben Nutzerinnen und Nutzern mehr Kontrolle und garantieren die Unabhängigkeit, die Software (später) selbst oder durch Dienstleister pflegen und weiterentwickeln zu können.

Auch Hersteller, die Internetrouter, Fernseher, Car-Entertainment und andere **Hardwaresysteme** in großen Stückzahlen produzieren, nutzen heute fast ausschließlich Linux. Proprietäre Systeme wurden in spezialisierte Nischen zurückgedrängt. Jedes Unternehmen kann wechselseitig auf alle Innovationen in Linux zugreifen. Gerade darin zeichnet sich das Open-Source-Wesen aus, das Kooperation und Kollaboration

4 Vgl. dazu https://www.golem.de/news/open-Source-microsoft-hostet-mehr-linux-als-windows-vms-in-azure-1907-142245.html
5 Vgl. dazu https://github.com/readme/nasa-ingenuity-helicopter
6 Siehe https://www.bitkom.org/openSourcemonitor

sowie das Teilen der Ergebnisse hochhält: Technologien, die in einem offenen Prozess entwickelt werden, stehen allen zur Verfügung. Konsequenterweise muss ein Unternehmen das komplexe Ganze nicht mehr eigenständig im jeweils eigenen System aufbauen und pflegen. Stattdessen geben alle ein wenig und erhalten in der Summe das Ganze. Das reduziert die eigenen Entwicklungskosten, und der Wettbewerb verlagert sich in Richtung strategischer Spezialisierungen und Serviceangebote.

Diese Hinwendung zu einer gemeinsamen, kooperativen Bereitstellung der Technologie an sich und die Konzentration auf ein Alleinstellungsmerkmal, das sich in einem auf dieser Technologie aufbauenden Service manifestiert, zeigt sich sehr augenscheinlich in der Cloud: Die sogenannten Cloud-Native-Technologien rund um Kubernetes und andere Open-Source-Projekte sind prominente Technologien, auf die unterschiedliche Firmen aufbauen, wenn sie konkurrierende Angebote in den Markt bringen – obwohl sie zugleich die zugrunde liegende Technologie erfolgreich gemeinsam weiterentwickeln.

Damit geht einher, dass mit **Open-Source-Software** sehr wohl **Gewinn** erzielt werden kann, denn, dass Open-Source-Software nicht kommerziell verwendet werden darf, ist ein Irrglaube. Das Gegenteil ist der Fall: Es gibt eine Reihe von etablierten Geschäftsmodellen, die sich der Vorteile von Open-Source-Software bedienen. Immer wieder bilden sich auch neue Geschäftsmodelle, die von der Offenheit von Open Source und der enormen Reichweite, die mit Open-Source-Projekten erreicht werden kann, profitieren. Die Akquisition von RedHat durch IBM im Jahr 2019[7] ist als eine der größten Akquisitionen eines Software-Unternehmens in die jüngere Geschichte eingegangen. Das Geschäftsmodell von RedHat basiert ausschließlich auf Open-Source-Software.

> Dass Open-Source-Software nicht kommerziell verwendet werden darf, ist ein Irrglaube.

Allerdings können diese Geschäftsmodelle nicht darin bestehen, über Lizenzentgelte und Lizenzverträge Umsatz zu generieren. Denn auch im kommerziellen Kontext bleibt das Wesen der Open-Source-Software natürlich erhalten: Konstitutiv gehört hierzu, dass für die Kundinnen und Kunden **keine Lizenzkosten** anfallen dürfen. Für die Arbeit der Zusammenstellung und Installation von Software im Sinne eines Services darf allerdings ebenso ein Entgelt verlangt werden, wie für Support, Maintenance oder Betrieb, also generelle Leistungen, die mit dem Einsatz von Open-Source-Software zusammenhängen oder mit ihr erbracht werden.

> Mit Open-Source-Software verlagert sich das Geschäftsmodell weg vom reinen Lizenzgeschäft.

Mit Open-Source-Software verlagert sich das Geschäftsmodell weg vom reinen Lizenzgeschäft hin zu anderen Geschäftsmodellen mit und um Open-Source-Software herum, wie zum Beispiel der Subskription von Diensten (ausführliche Darstellung im ↗ Kapitel 5).

Dennoch ist der Einsatz von Open-Source-Software nicht zum Nulltarif zu bekommen, auch wenn die Verwendung selbst bzw. das Recht dazu – finanziell gesehen – in der Tat kostenlos ist – und zwar immer. Der Aufwand, um Software verantwortungsvoll einzusetzen und dabei sicherzustellen, dass unter anderem Lizenz-Compliance, Sicherheit, oder Support angemessen geregelt sind, schlägt sich in entsprechenden Kosten nieder. Es haben sich jedoch Standard-Vorgehensweisen herausgebildet, mit denen man diese Herausforderungen effektiv angehen kann.[8]

7 Siehe https://www.redhat.com/en/about/press-releases/ibm-closes-landmark-acquisition-red-hat-34-billion-defines-open-hybrid-cloud-future

8 Mehr dazu in den Kapiteln zu Strategie, Compliance und Open-Source-Management

2.2 Konzept und Definition von Open-Source-Software

Um Open-Source-Software effektiv einzusetzen, ist es hilfreich, zunächst das Konzept »Open-Source-Software« selbst zu verstehen, das sich auch in und mit einem gewissen Begriffsapparat manifestiert. Der Ursprung liegt in der Bewegung für Freie Software, die in den 1980er Jahren von der amerikanischen Free Software Foundation begründet wurde. Sie formuliert als **Kernidee** die vier fundamentalen Freiheiten, die jede Software erfüllen muss, wenn sie Freie und Open-Source-Software sein will: Wer auch immer sie hat, muss zugleich auch das Recht haben,

- sie (a) **auszuführen**,
- sie (b) zu **analysieren**,
- sie (c) an die eigenen Bedürfnisse **anzupassen** und
- sie (d) **weiterzugeben**, auch in veränderter Form.[9]

Eine notwendige Voraussetzung für die Ausübung der vier Kernrechte ist der Zugriff auf den Quellcode. Dies spiegelt sich im Begriff »Open Source« wider, der sich inzwischen, insbesondere im geschäftlichen Umfeld, als der geläufige Begriff etabliert hat, um Software zu bezeichnen, die diese vier fundamentalen Freiheiten erfüllt[10]. Die im Jahr 1998 gegründete Open-Source-Initiative (OSI)[11] hat daraus die zehn Kriterien umfassende **Open-Source-Definition** (OSD) entwickelt[12]. Sie ist heute allgemein anerkannt, um zu klassifizieren, welche Lizenzen als Open-Source-Lizenzen betrachtet werden. Neben dem Recht zur Weitergabe der Software und der Verfügbarkeit von Quellcode stellen die Kriterien der Open-Source-Definition insbesondere die diskriminierungsfreie Verwendung von Open-Source-Software sicher, so dass diese nicht auf bestimmte Anwendungsfälle, Personengruppen oder technologische Bereiche und Produkte eingeschränkt werden kann.

Die für Open-Source-Software **konstitutiven Rechte** der **Nutzung**, **Einsicht**, **Veränderung** und **Distribution** (auch in veränderter Form) sind nicht per se mit Software verbunden: Software unterliegt dem Urheberrecht.[13] Danach stehen sämtliche Rechte an einer Software zunächst deren Urhebern zu. Nur sie können bestimmen, was andere mit dem von ihnen geschaffenen Werk tun dürfen und was nicht. Damit andere

9 vgl. die Begriffsklärung der Free Software Foundation Europe zu freier Software http://fsfe.org/about/basics/ freesoftware.de.html

10 Neben »Open-Source-Software« und »Freier Software« wird manchmal auch der zusammengezogene Begriff »Free and Open-Source-Software (FOSS)« oder »Free, Libre, and Open-Source-Software (FLOSS)« verwendet. Anhängerinnen der Bewegung für Freie Software betonen, dass die Verwendung des Begriffs »Frei« wichtig ist, um die ideellen Werte, die hinter den Freiheiten stehen, zu betonen. In der Praxis, insbesondere aus Lizenzsicht, besteht aber nur ein sehr kleiner Unterschied zwischen »Freier Software« und »Open-Source-Software«. Wir verwenden deshalb in diesem Leitfaden durchgängig den gebräuchlichen Begriff »Open Source«.

11 vgl. http://openSource.org/

12 vgl. http://openSource.org/osd

13 vgl. z. B. für Deutschland §§ 69a ff. des Urheberrechtsgesetzes (UrhG)

Personen als die Urheber eine Software in rechtmäßiger Weise einsetzen können, muss ihnen der Urheber eine **Nutzungsberechtigung** einräumen. Für eine solche Nutzungsberechtigung, die regelmäßig mit bestimmten Nutzungsbedingungen und/oder Nutzungsbeschränkungen verbunden ist, hat sich der Begriff **»Lizenz«** eingebürgert. Will also ein Urheber – oft die Programmiererinnen und Programmierer oder ihre Arbeitgeber – eine Software als Open-Source-Software vertreiben, muss er die genannten Freiheitsrechte explizit einräumen. Er muss sein Werk unter eine freie Lizenz stellen, unter eine Open-Source-Lizenz.

Die OSI ist außerdem über die Festlegung der Definition hinausgegangen und hat einen **»License Review Process«** mit dem Ziel eines »Approvals« implementiert.[14] Damit unterliegt es nicht der persönlichen Interpretation, ob es sich im Einzelfall wirklich um Open-Source-Software bzw. um eine Open-Source-Lizenz handelt. Für über 60 Open-Source-Lizenzen hat die OSI sichergestellt, dass sie die Kriterien der Open-Source-Definition erfüllen und listet sie offiziell.[15]

Dies hat einen praktischen Wert: Wer Software einsetzt, die unter einer offiziell gelisteten Lizenz freigegeben worden ist, weiß auch ohne Blick in die einzelne Lizenz, dass sie im Hinblick auf diese Software alle Rechte hat, die die OSD zur Voraussetzung macht. Der Anwender weiß außerdem, dass er keine Einschränkungen berücksichtigen muss, die die OSD ausschließt. Dies gilt allerdings nur, soweit es sich um einen von der OSI freigegebenen, unveränderten Lizenztext handelt. Welche Bedingungen an die Ausübung der Rechte aus einer Lizenz gekoppelt sind, offenbart nur die jeweilige Lizenz selbst. Jedenfalls ist Open-Source-Software letztlich nur dann wirklich Open-Source-Software, wenn sie unter einer offiziellen Open-Source-Lizenz freigegeben worden ist, die die zehn Kriterien der OSD erfüllt und die von der OSI bestätigt ist.

Welche Bedingungen an die Ausübung der Rechte aus einer Lizenz gekoppelt sind, offenbart nur die jeweilige Lizenz selbst.

Noch offen ist die Frage, wie Software zu bezeichnen ist, die nicht Open-Source-Software ist. Geläufig sind die Begriffe **»proprietäre«** oder **»kommerzielle Software«**. Der Begriff »kommerzielle Software« ist jedoch irreführend, da auch Open-Source-Software kommerziell eingesetzt werden kann und dies auch in großem Maßstab geschieht. Es existieren zudem viele Geschäftsmodelle, bei denen Open Source ein wesentlicher Teil der kommerziellen Strategie ist. Im Folgenden wird deshalb den Begriff »proprietäre Software« verwendet, um auszudrücken, dass sich der Rechteinhaber wesentliche Rechte vorbehält und Nutzerinnen und Nutzer nicht die vollen vier Freiheiten zur Verfügung stellt.

Auch wenn die rechtliche Beurteilung einer Software nicht von der Benennung, sondern von den jeweiligen Nutzungsbedingungen abhängt, hat sich ein Diskurs über Lizenztypen etabliert. Schon der Begriff »Open-Source-Software« selbst verallgemeinert – wie jeder Begriff. Außerdem begegnet man in der Praxis gelegentlich einer gewissen laxen Sprechweise. Dies kann bis zur absichtlichen Irreführung gehen, wo versucht wird, von der positiven Belegung des Begriffs »Open Source« zu profitieren, auch wenn gar keine Open-Source-Software angeboten wird.

14 vgl. Open-Source-Initiative: The License Review Process http://openSource.org/approval
15 vgl. Open-Source-Initiative: Licenses by Name http://openSource.org/licenses/alphabetical

Dies läuft unter dem Etikett »Open Washing«. Wer angemessen am Diskurs teilnehmen will, sollte den Terminus »Open-Source-Software« nur verwenden, wenn die in Rede stehende Software unter einer von der OSI offiziell als Open-Source-Lizenz anerkannten Lizenz veröffentlicht wird. Eine missbräuchliche Falschbezeichnung als Open-Source-Software könnte als Verstoß gegen das Wettbewerbsrecht gewertet werden.

In diesem Zusammenhang ist auch ein Lizenz-Typ zu nennen, der in den letzten Jahren häufiger aufgetaucht ist, und der als »Source Available« bezeichnet werden kann. Lizenzen dieses Typs, die manchmal als Änderung weg von einer anerkannten Open-Source-Lizenz entstanden sind, bieten weitgehend die gleichen Rechte wie eine Open-Source-Lizenz, unter anderem auch die Verfügbarkeit des Quellcodes, schließen aber spezifische Nutzungsszenarien aus.[16] Damit verletzen sie die erste fundamentale Freiheit und das Diskriminierungsverbot der Open-Source-Definition. Somit sind sie keine Open-Source-Lizenzen und müssen als proprietäre Lizenzen im Einzelfall analysiert und betrachtet werden.

16 Beispiele sind die Server Side Public License (SSPL) oder der Commons Clause, die versuchen zu verhindern, dass Public Cloud Provider die Software als Service anbieten.

2.3 Chancen und Herausforderungen

Chancen und Herausforderungen von Open-Source-Software zu thematisieren, heißt zu werten. Denn Vor- und Nachteile sind insofern immer subjektiv, als sie immanent Vor- oder Nachteile »für jemanden« sind. Dennoch können auch Personen, die dem Phänomen »Open-Source-Software« ob der eigenen Vorteile ganz grundsätzlich gewogen gegenüberstehen, die Lage insgesamt differenziert betrachten. Die Welt ist selten »schwarz-weiß«. Und was Chancen bietet, kann an anderer Stelle immer noch mit Risiken einhergehen. In diesem Sinne wollen die Autorinnen und Autoren, die sich gerne und offen als ein Team von »Open-Source-Befürworterinnen und -Befürwortern« verstehen, es dennoch wagen, ein gewiss subjektives, aber eben auch rundes Fazit zu ziehen.

2.3.1 Chancen

Open-Source-Software bietet Vorteile, die ihr immanent sind – erst durch die damit verbundenen Eigenschaften zeichnet sie sich überhaupt als Open-Source-Software aus:

Vier Open-Source-Software-Grundrechte: Open-Source-konforme Lizenzen räumen den Nutzerinnen und Nutzern der so lizenzierten Software weitreichende Rechte ein. Die wichtigsten sind die »4 Grundrechte«, nämlich die Software nutzen zu dürfen, und zwar für jeden Zweck, sie untersuchen zu dürfen, sie verteilen zu dürfen und sie verändern zu dürfen und sie in veränderter Form weitergeben zu dürfen.17 Die daraus resultierende volle Kontrolle über den Source Code führt zu zwei weiteren wichtigen Punkten:

Transparenz: Gut organisierte Open-Source-Software-Projekte haben neben einem für alle zugänglichen Source-Code-Control-System wie Git oder Subversion unterschiedliche Code-Stränge für stabile Releases und die laufenden Entwicklungen, klare Releases, öffentlich verfügbare Mailinglisten, Bugtrackingsysteme, Wikis und so weiter. Ein gutes Open-Source-Software-Projekt wird mithin in der Öffentlichkeit entwickelt, nicht hinter verschlossenen Türen. Das diese Informationen für alle zugänglich sind, bedeutet jedoch nicht, dass jeder, der möchte, schreibend auf das Source-Code-Control-System zugreifen kann. In vielen Open-Source-Software-Projekten haben nur wenige Personen direkte Mitwirkungsrechte. Zwar können alle mit Änderungen zum Projekterfolg beitragen. Aber oft werden diese »Vorarbeiten« überprüft und nur nach Begutachtung von gesonderten Integratorinnen und Integratoren in die Projektquellen

> Ein gutes Open-Source-Software-Projekt wird mithin in der Öffentlichkeit entwickelt, nicht hinter verschlossenen Türen.

17 vgl. https://www.gnu.org/philosophy/free-sw.html

aufgenommen. So wird gewährleistet, dass nur Code in den Source-Tree des Projektes aufgenommen wird, der vorher durch einen »Peer Review« gegangen. Dies sichert zum einen die Qualität. Zum anderen gewährleistet das Verfahren, dass jede Codesequenz einer Person zugeordnet werden kann. Dies ist besonders bei Lizenzierungsfragen und urheberrechtsrelevanten Aktionen wichtig.

Eine solche Transparenz steigert das Vertrauen in Software, was insbesondere bei Software, die mit sensiblen Daten umgeht, ein notwendiger Erfolgsfaktor ist.[18]

Qualität: Open-Source-Software zeichnet sich häufig durch hohe Qualität aus. Die transparenten Entwicklungsprozesse ermöglichen diese Qualität durch die Diversität der Entwickler-Community und deren Interaktion mittels Peer-Reviews von Konzepten und Code. Zur Steigerung der Effizienz sind Open-Source-Projekte meist auch Vorreiter bei der Automatisierung von Test-, Build- und Release-Prozessen. Dies trägt einen wesentlichen Anteil zur Qualität der Software bei.

Erhöhte Innovationsgeschwindigkeit: Viele Open-Source-Software-Pakete realisieren sehr schnell neue Leistungsmerkmale und stellen an sich eine innovative Lösung dar. Dies mag für manche Softwarehersteller ein Nachteil sein, sofern sie mit ihrer proprietären Software in Konkurrenz zu den Open-Source-Software-Paketen stehen. Für Hersteller, die diese innovativen Open-Source-Software-Pakete als Bestandteil eigener Produkte integrieren, wiegt der Vorteil aber doppelt. Denn zum einen sind auf diese Weise innovative Leistungsmerkmale in dem Produkt schnell verfügbar und zum anderen werden durch die Entlastung der eigenen Entwicklerinnen und Entwickler und des Projektbudgets Ressourcen frei, die in die Entwicklung weiterer Unique Selling Points (USPs) des Produktes investiert werden können.

Weiterverwendbarkeit: Open-Source-Software erhöht nicht nur die Innovationsgeschwindigkeit, sondern kann selbst Inkubator für Innovationen sein. So können bestehende Open-Source-Software-Lösungen Baustein oder Basis für weitere Open-Source-Software-Produkte sein. Auch hier können Ressourcen geschont bzw. freigesetzt und so in die eigentliche Innovation gesteckt werden.

Open-Source-Software erhöht nicht nur die Innovationsgeschwindigkeit, sondern kann selbst Inkubator für Innovationen sein.

Schnellerer Release: Produkte, die Open-Source-Software enthalten, können schneller auf den Markt gebracht werden. Der Einsatz von Open-Source-Software entlastet die eigenen Entwicklerinnen und Entwickler und das Projektbudget. Außerdem sind gut gepflegte Open-Source-Software-Pakete in der Regel auch gut getestet und weisen eine höhere Testabdeckung auf als die selbst entwickelten Alternativen. Open-Source-Software zu verwenden, trägt in der Regel also dazu bei, Produktstabilität zu erreichen.

18 Als Beispiel sei die Corona-Warn-App genannt, die als transparentes Open-Source-Projekt entwickelt worden ist, um die Akzeptanz für die Nutzung in der Bevölkerung zu erhöhen. https://github.com/corona-warn-app

Dies liegt auch daran, dass das Einsatzspektrum von Open-Source-Software prinzipiell breiter ausgelegt ist. Selbst entwickelte Lösungen sind oft spezifisch auf ein Produkt oder auf das Tätigkeitsfeld des Unternehmens zugeschnitten. Werden die breiter angelegten Eigenschaften der Open-Source-Software geschickt ausgenutzt, können Produkte dadurch früher auf den Markt gebracht werden.

Unabhängigkeit vom Hersteller: Nutzerinnen und Nutzer von Open-Source-Software-Paketen haben in der Regel einen größeren Handlungsspielraum. Fehler (Bugs) können auf den Mailinglisten gemeldet werden – oftmals werden »gut« gemeldete Fehler schnell von der Entwicklergemeinde behoben. Allerdings haben Meldende keinen Anspruch auf die Beseitigung der Fehler (siehe Risiken). Stattdessen können eigene Entwicklerinnen und Entwickler oder explizit beauftragte Software-Dienstleister den Fehler beseitigen oder die Erweiterung beitragen. Gleiches gilt auch für die Wartung. Außerdem können beauftragte Serviceerbringer leichter ersetzt werden, da die Basis ihres Tuns, die Software, frei zur Verfügung steht.

Lizenzgebührenfreiheit: Um Open-Source-Software einzusetzen, müssen und dürfen keine Lizenzgebühren gezahlt oder gefordert werden.

> Um Open-Source-Software einzusetzen, müssen und dürfen keine Lizenzgebühren gezahlt oder gefordert werden.

Als vorteilhafte Aspekte im Unternehmen ergeben sich:

Leichte Integration und Anpassbarkeit: Dadurch, dass der Source Code der Open-Source-Software-Pakete verfügbar ist, lassen sich diese leicht in die meist heterogene IT-Landschaft eines Unternehmens integrieren und/oder für den spezifischen Einsatz in einem Produkt anpassen. Open-Source-Software-Pakete sind sozusagen Rohdiamanten, die für den Einsatz in Unternehmen und Produkten nur noch geschliffen werden müssen.

Kompetenzausbau der eigenen Mitarbeitenden: Eigene Mitarbeitende können durch die Analyse des Source Codes die eigene Programmierkompetenz erhöhen. Viele prominente Open-Source-Software-Projekte haben Referenzcharakter in Bezug auf die Lösung von programmiertechnischen Herausforderungen. Demzufolge können die in den Projekten realisierten Lösungen als Vorlagen für die Lösung ähnlicher Fragestellungen herangezogen werden.

> Viele prominente Open-Source-Software-Projekte haben Referenzcharakter in Bezug auf die Lösung von programmiertechnischen Herausforderungen.

Firmenübergreifende Möglichkeit zur Standardisierung: Ein Großteil der in Produkten genutzten Software hat keinen differenzierenden Faktor, wird aber benötigt, um die Funktionalität des Produkts zu gewährleisten. Gerade diese Software bildet die Grundlage für firmenübergreifende Defacto-Standards, die in Kooperation mit anderen Marktteilnehmern als Open-Source-Software entwickelt werden. Diese Herangehensweise ermöglicht es allen, die Aufwände für diese nicht wettbewerbsrelevanten Anteile zu senken und trotzdem eine stabile Basis für die Produktentwicklung zu schaffen. Die frei gewordene Entwicklungskapazität kann für differenzierende Funktionalitäten eingesetzt werden, was die Marktchancen der beteiligten Firmen stärkt.

Durch die Mitarbeit an der Open-Source-Software wird aber auch der Einfluss der beteiligten Firmen an der Software gewährleistet, die Firma bleibt »am Puls der Zeit« und vermeidet das Risiko, von Änderungen überrascht zu werden. Das verhindert nicht kalkulierte Aufwände zur Behebung von strukturellen Diskrepanzen zwischen der Open-Source-Basis und den eigenen, proprietären Softwareanteilen.

2.3.2 Herausforderungen

Sicherheit: Es bleibt Open-Source-Software-Nutzerinnen und -Nutzern nicht erspart, sich mit einer besonderen Diskrepanz auseinanderzusetzen: Wer mit Open-Source-Software umgeht, begegnet bald der Frage, wie denn eine quelloffene Software Sicherheit gewährleisten kann, wo doch alle Möglichkeiten des Missbrauchs und Ausnutzens von Programmierfehlern dem Konzept nach frei und offen zur Verfügung gestellt werden. Daran gibt es nichts zu deuteln:

Bei Open-Source-Software liegen die Dinge – dem Wesen von Open-Source-Software nach – wirklich offen zu Tage, und zwar offener, als bei proprietärer Software, bei der der Quellcode unter Verschluss bleibt. Daraus zu schließen, dass Open-Source-Software unsicherer sei als proprietäre Software und dass proprietäre Software darum vom Konzept her immanent sicherer sei, ist ein Irrtum. Denn die Nutzerinnen und Nutzer von proprietärer Software wissen schlicht nicht, welche Möglichkeiten des Missbrauchs und Ausnutzens von Programmierfehlern darin enthalten sind. Erst recht wissen sie nicht, ob diese Missbrauchsmöglichkeiten – absichtlich oder unabsichtlich – von den Herstellern nicht längst schon an Dritte weitergereicht wurden. Aus der Tatsache, dass etwas nicht zu sehen ist, zu schließen, dass es nicht da ist, ist illusionär. Die Nutzerinnen und Nutzer proprietärer Software sind in dieser Hinsicht auf Zusicherungen der Lieferanten angewiesen.

Ganz anders die Open-Source-Software-Nutzerinnen und -Nutzer. Gerade weil alle Quellen offen liegen, können sie – wenn es ihnen wichtig ist – die Sicherheit der Software selbst überprüfen. Und wenn es ihnen nicht so wichtig ist, können sie – wenigstens bei vielfach genutzter Open-Source-Software – immer noch darauf vertrauen, dass andere diese Möglichkeit der Kontrolle tatsächlich genutzt haben. So zeigt sich denn auch, dass bei quelloffener Software – ausgelöst durch die Community – die Verbesserungen und Korrekturen oft viel kürzer getaktet zur Verfügung gestellt werden, als bei (kommerziell getriebener) proprietärer Software. Mithin stellt sich der vermeintliche Nachteil bei näherem Hinsehen sogar als Vorteil heraus.

Bei der Vielzahl eingesetzter Open-Source-Module, deren Zahl leicht in die Tausende gehen kann, entsteht noch eine weitere Herausforderung: Wie ist zu verhindern, dass Schwachstellen in einem der vielen Module übersehen werden? Hier gibt es im wesentlichen zwei Gefahren: Gezielte Angriffe durch Einschleusen manipulierter Module mit Schadcode, sogenannte Supply-Chain-Attacks und das Verwenden veralteter Versionen mit bekannten Schwachstellen. Mit geeigneten Prozessen und Werkzeugen zur Automatisierung kann diesen Gefahren begegnet werden.

Open-Source-Lizenzen bieten weitgehende Freiheiten, verknüpfen diese aber mit einigen Pflichten.

Lizenz-Compliance: Open-Source-Lizenzen bieten weitgehende Freiheiten, verknüpfen diese aber mit einigen Pflichten. So müssen in der Regel der Lizenztext und Urheberrechtshinweise mit der Software ausgeliefert werden. Einige Lizenzen fordern zudem, auch den Quellcode zur Verfügung zu stellen, manche generell, andere wenigstens in Fällen von Änderungen. Solche Bedingungen zu erfüllen, stellt in der Regel keine große Hürde dar, bedarf aber einer sorgfältigen und vollständigen Behandlung aller verwendeten Open-Source-Software.

Eine spezifische Herausforderung für die Open-Source-Software-Nutzerinnen und -Nutzer betrifft die Möglichkeit, dass durch die Verwendung von Open-Source-Software in vermarkteten Softwareprodukten eigenes »Core-Know-how« wegen des mit der Open-Source-Software-Lizenz etablierten Copyleft-Effekts offengelegt werden muss (Verweis auf ↗ Kapitel 7.2). Geschäftskritische Alleinstellungsmerkmale können auf diese Weise ungewollt »generalisiert« werden. Dass diese Möglichkeit besteht, ist zunächst einmal ein Faktum. Gleichwohl stellt sich auch diese »Gefahr« bei näherem Hinsehen als nicht allzu brennend dar: Meistens ist nämlich das, was das eigene Geschäft im Vergleich mit Konkurrenten werthaltig als Alleinstellungsmerkmal begründet, vom Umfang gering – auch wenn es als objektive »Kleinigkeit« subjektiv von großer Bedeutung ist. Das bedeutet umgekehrt, dass der größere Teil des eigenen Geschäftes sehr wohl auch in und mit Copyleft behafteter Open-Source-Software realisiert werden kann, ohne Konkurrenten einen ungewollten Vorteil zu bieten. Zudem ist zu berücksichtigen, dass sich der Copyleft-Effekt nur in bestimmten Einsatzszenarien und technischen Architekturen auf eigenen Code erstreckt. Es ist deshalb wichtig, genau zu differenzieren.

Support: Für den zuverlässigen und sicheren Betrieb von Software muss gewährleistet werden, dass der notwendige Support besteht, um kritische Fehler und Sicherheitsprobleme in angemessener Zeit beheben zu können. Das wird üblicherweise durch Software-Hersteller als kommerzielle Leistung angeboten.

Bei Open-Source-Software ergeben sich hier zusätzliche Möglichkeiten. Zum einen können auch andere Anbieter beauftragt werden, da durch die Offenheit des Codes nicht nur einzelne Hersteller die Kompetenz und Rechte haben den Code zu pflegen und damit entsprechende Support-Leistungen anbieten zu können, sondern dies auch durch Dritte erfolgen kann. Zum anderen können auch die Nutzerinnen und Nutzer selbst in diese Rolle schlüpfen und den Support eigenständig leisten. Dabei ist oft die Community, die hinter dem Projekt steht, eine große Hilfe. Es muss jedoch genau abgewogen werden, welche Garantien im konkreten Einsatzfall nötig sind, und wer diese zuverlässig und im erforderlichen Zeitrahmen leisten kann.

Open-Source-Lizenzen geben zwar weitgehende Rechte an dem veröffentlichten Stand des Codes, schließen aber Haftung und Gewährleistung aus.

Nachhaltigkeit von Open-Source-Projekten: Mit der Veröffentlichung von Open-Source-Software geht man keine formale Verpflichtung ein, diese Software auch dauerhaft zu pflegen. Open-Source-Lizenzen geben zwar weitgehende Rechte an dem veröffentlichten Stand des Codes, schließen aber Haftung und Gewährleistung aus – jedenfalls so weit es der nationale Rechtsrahmen erlaubt. Außerdem legen sie nichts hinsichtlich Zeitpunkt, Art und Umfang zukünftiger Entwicklung fest, außer u. U., dass

bei manchen Lizenzen zukünftiger Code wiederum unter der gleichen Lizenz veröffentlicht werden wird.

Dies kann zu Problemen auf Seiten der Nutzerinnen und Nutzer führen, die Open-Source-Software über einen längeren Zeitraum einsetzen und damit darauf angewiesen sind, dass Open-Source-Projekte auch nachhaltig entwickelt werden. Zwei Aspekte sind hier besonders zu berücksichtigen, die **Governance** von Projekten[19], sowie ihre **Finanzierung**. Als verantwortungsvoller Nutzer von Open-Source-Software muss man dies sowohl bei der Auswahl von Projekten berücksichtigen, als auch bei verwendeten Projekten Wege finden, zur Nachhaltigkeit der Projekte beizutragen und sich damit die Grundlage für eine zuverlässige Pflege der Projekte zu erhalten.

2.3.3 Fallstricke

Schließlich gibt es Herausforderungen, die manchmal als spezielle »Probleme« von Open-Source-Software dargestellt werden, obwohl sie doch in gleichem oder ähnlichem Sinne auch proprietäre Software betreffen. Wir listen einige dieser Aspekte auf:

Lizenzrechtliche Verwicklungen: Softwarepakete sind heute komplex. Sie enthalten Komponenten, die nicht notwendigerweise unter der Hauptlizenz weitergegeben werden. In einer Zeit, in der immer mehr kommerzielle Produkte auch im Verbund mit Open-Source-Komponenten vertrieben werden, entsteht diese Herausforderung bei Open Source- und proprietärer Software gleichermaßen.

IPR-Seiteneffekte: Open-Source-Software kann – auch ungewollt – IPRs (Intellectual Property Rights) Dritter verletzen. Programmiererinnen und Programmierer können ganz ohne Absicht Patente benutzen, ohne Patentlizenzen zu erwerben. Allerdings gilt wiederum, dass dies Open-Source- und proprietäre Closed-Source-Software gleichermaßen betrifft: Beide Arten der Software können die (Patent)rechte anderer verletzen, sodass die Nutzerinnen und Nutzer der Software der bloßen Nutzung wegen deswegen angegangen wird. Bei Software, die kommerziell von einem Hersteller bezogen wird, glaubt man allerdings, in und mit der Gewährleistung des Lieferanten einen Anspruchsgegner für derartige mittelbare Schäden zu haben. Ob die finanzielle Kraft des Lieferanten ausreicht, diesen Schaden samt zugehörigem juristischen Aufwand tatsächlich abzufedern, bedarf einer gesonderten Überlegung.

> Jede Verletzung der Überlassungspflichten birgt die Gefahr des dauerhaften Entzugs der Nutzungsrechte.

Entzug von Nutzungsrechten: Jede Verletzung der Überlassungspflichten birgt die Gefahr des dauerhaften Entzugs der Nutzungsrechte. Dies kann bei proprietärer Software ebenso geschehen wie bei Open-Source-Software.

19 Vergleiche auch den Abschnitt zu Open-Source-Foundations, die zur Nachhaltigkeit von Projekten durch Etablieren einer »Open Governance« beitragen.

Geringerwertige Haftungszusagen: Open-Source-Software-Lizenzen enthalten immer einen sogenannten Haftungsausschluss. Juristischer Besonderheiten wegen reduziert sich dieser Ausschluss in Deutschland – sehr grob gesagt – auf die Haftungszusagen, die mit einer Schenkung einhergehen. Kommerziell vertriebene Software unterliegt stärkeren Verpflichtungen. Gleichwohl reduzieren sich auch deren Haftungsmöglichkeiten von der geschäftlichen Kraft her, während umgekehrt kommerzielle Distributoren von Open-Source-Software als eines ihrer Geschäftsmodelle gerade die originäre begrenzte Open-Source-Software-Haftung aufwerten.

Unbekannte Folgekosten: Nicht abgeschätzte und nicht geplante Kosten, wie upgrade- oder sicherungsbedingte Reintegrationsarbeiten, die durch die Verwendung von Open-Source-Software entstehen, können den Business Case eines Produktes in Frage stellen. Nur gilt sicher auch für proprietäre Software, dass nicht abgeschätzte und nicht geplante Kosten den Business Case eines Produktes korrumpieren. Was aber sicher
(für beide) nicht gilt, ist, dass notwendigerweise nicht abschätzbare und nicht planbare Kosten auf die Nutzerinnen und Nutzer zukommen. Hier wie da bedarf es eben einer handwerklich sauberen Analyse der Folgen, bei der Verwendung von proprietärer Software ebenso wie bei der Nutzung von Open-Source-Software.

Know-How-Defizite: Die einfache Verwendung von Open-Source-Paketen verführt manchmal dazu, den Aufwand für die Integration in das eigene Produkt zu unterschätzen und für jederzeit reproduzierbar zu halten. Allerdings setzt diese Arbeit Fachwissen voraus. Was bei kommerzieller Nutzung sozusagen »automatisch« im Service mitgeliefert und vergütet wird, muss im Hinblick auf direkt vom Open-Source-Projekt bezogene Pakete auch organisiert werden. Verzichtet man darauf, kommt das dem Verzicht auf den Hersteller-Support gleich.

Die einfache Verwendung von Open-Source-Paketen verführt manchmal dazu, den Aufwand für die Integration in das eigene Produkt zu unterschätzen.

Veraltete Software: Open-Source-Software kann veralten, sei es in der Entwicklung, sei es in der Dokumentation, sei es in der Distribution. Plötzlich stehen die Nutzerinnen und Nutzer vor »toten«, nicht mehr gepflegten Paketen. Allerdings gibt es auch auf proprietärer Seite den ähnlichen Fall. Auch Firmen können »veralten«. Sie können schließen, ihren Fokus verändern oder aufgekauft werden. Und ebenso plötzlich stehen auch die Kundinnen und Kunden proprietärer Software vor »toter« Software. Auf Open-Source-Seite gibt es aber – der vier Freiheiten wegen – wenigstens die prinzipielle Möglichkeit einer »Wiederbelebung«.

Negative Presse: Im Falle von Open-Source-Software-Lizenzverletzungen gibt es durchaus die Möglichkeit, dass die Softwareszene kommunikativ heftig reagiert und so den persönlichen Ruf schädigt. Umgekehrt stehen den Lieferanten proprietärer Software bei Verletzung ihrer Vertragsbedingungen ebenfalls heftige Mittel zu Verfügung, nur dass diese wohl in erster Linie juristisch sein dürften.

Fehlerbehebung: Es besteht gegenüber der Community kein Anspruch auf Fehlerbehebung.

In vielen Fällen wird man dennoch hervorragenden Support durch die Community erhalten und zusätzlich wird oft auch Support mit garantierter Verlässlichkeit gegen Vergütung angeboten.

Diese wenigen Beispiele zeigen, dass es durchaus auch bei Open-Source-Software Herausforderungen gibt. Sie zeigen allerdings genauso, dass auch auf der Seite von proprietärer Software entsprechenden Herausforderungen gemanagt werden müssen. In Sachen »Herausforderungen« ist Open Source nicht ganz so besonders, wie gelegentlich behauptet; in Sachen »Chancen« aber sehr wohl.

3 Nutzen von Open-Source-Software

3.1 Strategiebeispiele für den Einsatz von Open-Source-Software

Die Verwendung von Open-Source-Software in der eigenen IT sollte durchdacht sein. Im Laufe dieses Leitfadens zeigen wir verschiedene Nutzen und Fallstricke von Open Source auf und widmen einer übergreifenden Open-Source-Strategie auch ein gesamtes Kapitel.

Wer Open-Source-Software aber nur nutzen möchte und sich noch keine Gedanken über das »Große Ganze« macht, findet in diesem Abschnitt Denkanstöße und Beispiele dazu, wie Open Source Einzug in das eigene Unternehmen finden kann.

3.1.1 Generelle Ablehnung

Das Verwenden von Open-Source-Software-Komponenten ist im Unternehmen nicht erlaubt.

Offenkundige Vorteile	Offenkundige Nachteile
Abschottung gegenüber unbekannten Vertriebs-, Kooperations-, und Lizenzmodellen.	Keine Nutzung von frei verfügbaren Gütern, Implementierungs- und Kontrollkosten der Strategie liegen nicht unbedingt unter den Kosten einer Strategie »pro« Open-Source-Software.
	Open-Source-Software ist Bestandteil der meisten proprietären Soft- und Hardwareprodukte. Als Nutzer auf Open-Source-Software zu verzichten, ist dadurch sehr aufwändig und in der Regel unmöglich.

3.1.2 Keine Open Source in eigenen Produkten

Das Einbringen von Open-Source-Software in die Lieferkette (eingebettet in eigene Produkte) ist nicht erlaubt.

Offenkundige Vorteile	Offenkundige Nachteile
Strikte Vermeidung unfreiwilliger Veröffentlichung von Alleinstellungsmerkmalen.	Nutzung aller frei verfügbarer Güter in der Eigenentwicklung ausgeschlossen.
Nutzung von Open Source durch Zulieferer möglich, wodurch Risiken, Support und Wartungsaufwendungen auf diese verschoben werden können.	

3.1.3 Nur ausgewählte Open-Source-Lizenzen in eigenen Produkten

Das Einbringen von Open-Source-Software in die Lieferkette ist möglich, sofern die Auflagen der jeweiligen Lizenzen und Wartungsaspekte vom Unternehmen **pauschal** akzeptiert werden. Typisch: Keine Komponenten mit strengem Copyleft.

Offenkundige Vorteile	Offenkundige Nachteile
Feingranularere Nutzung von Open-Source-Komponenten möglich.	Unterschiedliche Produkt- und Risikogruppen verlangen unterschiedliche Freigabeklassen. Eine einfache Allow-/Deny-List stößt hier oft an ihre Grenzen.
	Überwachung der Komponenten notwendig.

3.1.4 Ausgewählte Open-Source-Lizenzen in ausgewählten Produkten

Das Einbringen von Open-Source-Software in die Lieferkette ist möglich, sofern die Auflagen der jeweiligen Lizenzen und Wartungsaspekte vom Unternehmen **fall- oder Use-Case bezogen** akzeptiert werden. Typisch: Kein Copyleft in Embedded Devices und nur dann, wenn der Copy-Left-Effekt durch verschiedene Faktoren nicht zum Einsatz kommt.

Offenkundige Vorteile	Offenkundige Nachteile
Korrekteste Nutzung der Open-Source-Komponenten möglich.	Einzelfallprüfungen, Personalschulungen und feingranulare Modelle notwendig.
	Fehlentscheidungen möglich und Aufwände oft hoch.
	Detaillierte Überwachung der Komponenten notwendig.

3.1.5 Generelle Akzeptanz von Open Source in eigenen Produkten

Die Nutzung von Open-Source-Software ist erlaubt.

Offenkundige Vorteile	Offenkundige Nachteile
Voller Zugriff auf den gesamten Pool von Open-Source-Software.	Die reine Nutzung ist in den meisten Fällen keine nachhaltige Strategie.
Volle Kooperation mit der Community ist möglich.	Die Kooperation mit der Community bleibt unsystematisch.
	Es besteht die Gefahr, in die »Wartungsfalle« zu tappen: Gelegentlich wird Open-Source-Software noch modifiziert oder erweitert, bevor sie in die eigenen Produkte integriert wird. Werden diese Änderungen und Erweiterungen dann nicht an die entsprechenden Open-Source-Software-Projekte zurückgegeben, müssen sie nach einem Upgrade der Open-Source-Software-Basis erneut in den aktualisierten Softwarestand integriert werden. Das ist im Endeffekt »unproduktive« Arbeit.

3.1.6 Offensiv strategische Nutzung von Open Source in eigenen Produkten

Open-Source-Software ist von strategischer und wettbewerbsrelevanter Bedeutung. Das Unternehmen wird zu einem geschätzten und aktiven Teil des Open-Source-Ökosystems und seine Produkte und Services werden von den technischen und ökonomischen Vorteilen von Open-Source-Software profitieren.

Offenkundige Vorteile	Offenkundige Nachteile
Voller Zugriff auf den gesamten Pool von Open-Source-Software.	Für die Umsetzung der Strategie wird ein längerer Zeitrahmen zu veranschlagen sein, das heißt, dass es sich um ein »long term investment« handelt. Es muss allen Beteiligten bewusst sein, dass sich das Investment erst längerfristig richtig auszahlt. In diesem Zusammenhang sei noch einmal an die Definition von Strategie erinnert »ein mittelfristig oder langfristiges Unternehmensziel zu erreichen«. Demnach ist die Langfristigkeit kein Nachteil, sondern ein inhärentes Merkmal einer Strategie; das wird nur leider oft vergessen.
Volle Kooperation mit der Community wird systematisiert auf die Ziele des Unternehmens ausgelegt.	Die Strategie ist nur dann erfolgsversprechend, wenn ein Umdenkprozess in den Köpfen aller Beteiligten und Verantwortlichen stattfindet und die Bereitschaft besteht, eine gewisse Lernkurve zu durchschreiten. Des Weiteren muss man sich bewusst sein, dass Teile des Unternehmens dadurch möglicherweise transparenter werden. Das ist kein Nachteil an sich, man muss es nur vorab wissen und auch damit umgehen können.

3.2 Standardisierung und Kundenschutz

3.2.1 Zertifizierungen

Der Begriff »Open Source« selbst ist auch als Label erfolgreich und dient häufig als Marketinginstrument. Doch nicht alles, was als »Open Source« deklariert ist, verdient diesen Titel auch – zumindest nicht nach OSD oder der Definition der FSF für freie Software. Die Zertifizierung von Lizenzen durch die OSI (vgl. ↗ Abschnitt 2.2) erfüllt dabei eine wichtige Standardisierungsfunktion: Die Verwendung einer entsprechenden Lizenz garantiert Anbieterinnen und Anbietern sowie Nutzerinnen und Nutzern von Open-Source-Software Verlässlichkeit in Bezug auf die Einhaltung der jeweiligen Vorgaben der OSI.

Gelegentlich tritt jedoch Open-Source-Software in Erscheinung, die zwar auf Grundlage einer etablierten Lizenz aus der BSD-Lizenz-Familie, aber mit Abweichungen erstellt wurde, sodass das Ergebnis unter Umständen nicht mehr der OSD entspricht. Die betroffene Software kann nach wie vor quelloffen, dafür aber mit anderen Einschränkungen verbunden sein, zum Beispiel mit dem Verbot der kommerziellen Weiternutzung oder dem Ausschluss bestimmter Nutzergruppen (wie Rüstungsfirmen, Geheimdienste). Anbieter sollten darum auf eine von der OSI zertifizierte Lizenz setzen – zumal sich in der offiziellen Liste der OSI für fast jeden Anwendungsfall eine entsprechende Lizenz findet. Nutzerinnen und Nutzer sollten darauf achten, dass die Software unter einer der gelisteten Lizenzen steht. Für beide Seiten minimiert dies Unsicherheiten und potenzielle Risiken, entbindet aber nicht davon, die konkrete Nutzung im Einzelfall zu prüfen. Dies gilt besonders für ungewöhnliche Lizenzpflichten von nicht der OSD entsprechenden Lizenzen.

Mittlerweile sind Presse und Öffentlichkeit für Missbräuche des Labels »Open-Source-Software« sensibilisiert. Wer sich der Bezeichnungen »Open Source« oder »freie Software« bedient, sollte sich bewusst sein, dass Internet und soziale Medien schnell die Fälle aufdecken, in denen dies nur Fassade ist. Der potenzielle Image-Schaden kann höher sein als die Vorteile, die durch die Deklarierung erzielt werden sollten.

Zertifizierungen spielen im Bereich der Open-Source-Software eine nicht unerhebliche Rolle und leisten einen Beitrag zur Standardisierung. Zum einen können dabei die Zertifizierungen für (juristische) Personen betrachtet werden, zum anderen die für die eingesetzten Open-Source-Lizenzen. Aus der Analyse möglicher Open-Source-Geschäftsmodelle (vgl. ↗ Kapitel 5) ergibt sich, dass Unternehmen für ihre auf

Zertifizierungen spielen im Bereich der Open-Source-Software eine nicht unerhebliche Rolle und leisten einen Beitrag zur Standardisierung.

Open-Source-Software aufbauenden Angebote andere Kriterien benötigen, über die sie sich auszeichnen, als dies bei proprietären Ansätzen der Fall ist, die auf Lizenzentgelten basieren. Ausgewiesene Expertise ist ein solches Kriterium, das Vorteile gegenüber Mitbewerberinnen und Mitbewerbern schaffen kann. Die Qualifizierung der Mitarbeitenden wird durch die abgelegten Prüfungen am Linux Professional Institut (LPI)[20] deutlich. In zugehörigen Kursen (teils auch im Selbststudium) eignen sich die Interessierten Wissen verschiedener Erfahrungsstufen an und erhalten im Erfolgsfall ein Zertifikat. Mit dem LPI, den Lernmaterialien und den entsprechenden Prüfungen existiert also eine einheitliche, standardisierte Wissensbasis. Das erleichtert auch das Verfassen von Stellenangeboten sowie die Einschätzung des Kenntnisstandes einer Bewerberin oder eines Bewerbers. Bei Unternehmen steht z. B. ISO-5230[21] als Siegel der Open-Source-Software-Compliance.

3.2.2 Supportleistungen für Open-Source-Software

Open-Source-Software unterscheidet sich in puncto Supportleistungen in der Regel von proprietärer Software. Hauptgründe dafür sind zum einen eine nicht selten dezentrale Entwicklung der Software ohne konkrete Ansprechpartnerinnen und Ansprechpartner für die Nutzerinnen und Nutzer, zum anderen die meist kostenlose Überlassung der Software zur Nutzung oder Weiterentwicklung.

Während sich die Gewährleistung bei entgeltlich vertriebener Software nach den gesetzlichen Bestimmungen des Kauf- oder Werkvertragsrechts richtet, ist die Gewährleistung bei kostenlos überlassener Software wie Open-Source-Software eingeschränkt. Sie richtet sich im Regelfall nach den gesetzlichen Regelungen der Schenkung. Das hat zur Folge, dass der Hersteller für Mängel nur einstehen muss, wenn ihr Vorsatz oder grobe Fahrlässigkeit oder das arglistige Verschweigen eines Sach- oder Rechtsmangels vorzuwerfen ist. Ein Rechtsmangel liegt insbesondere vor, wenn der Nutzung der Software Rechte Dritter entgegenstehen. Dem Vorteil fehlender Lizenzkosten stehen daher eingeschränkte Gewährleistungsansprüche gegenüber.

> Um die Lücke fehlender Gewährleistungsansprüche zu schließen, kann es sich anbieten, entgeltliche Service- und Supportleistungen in Anspruch zu nehmen.

Um die Lücke fehlender Gewährleistungsansprüche zu schließen, die bei entgeltfreier Software entsteht, kann es sich anbieten, entgeltliche Service- und Supportleistungen in Anspruch zu nehmen. Die Palette solcher Angebote ist vielfältig – sie reicht von der Fehlerbehebung über das Schließen von Sicherheitslücken bis hin zur Weiterentwicklung und individuellen Anpassung der Software.[22]

Je nach Inhalt der vertraglichen Vereinbarung kommen verschiedene Vertragstypen im Hinblick auf Service- und Supportleistungen in Betracht: Sind die Leistungen nach der Vereinbarung der Parteien erfolgsbezogen ausgestaltet, wird es sich regelmäßig um einen Werkvertrag handeln. In diesem Fall wird ein konkretes Leistungsergebnis

20 vgl. https://www.lpi.org
21 https://de.wikipedia.org/wiki/ISO/IEC_5230
22 vgl. ↗Kapitel 5 zu Geschäftsmodellen

bzw. ein Tätigkeitserfolg geschuldet, so beispielsweise bei der Beseitigung von Störungen beziehungsweise Fehlern und der Aufrechterhaltung der Funktionsfähigkeit der Open-Source-Software. Geht es hingegen lediglich um die gewissenhafte Durchführung der Maßnahmen, beispielsweise um schlichte Beratungsleistungen beim Einsatz von Open-Source-Software, kommt ein Dienstvertrag in Betracht. Da bei diesem gerade kein fassbares Ergebnis geschuldet wird, sollte darauf geachtet werden, festzuschreiben, in welcher Qualität die Service- und Supportleistung zu erbringen ist.

Eine klare Einordnung als Werk- oder Dienstvertrag fällt nicht immer leicht und hängt letztlich von der Ausgestaltung der Vertragsbeziehung im Einzelnen ab. Die Beteiligten sollten darauf achten, die Vereinbarungen so eindeutig wie möglich zu gestalten, um auf diese Weise eine sichere Grundlage für die gemeinsame Zusammenarbeit zu schaffen und Unstimmigkeiten im Falle von Leistungsmängeln zu vermeiden. Ist ein Werkvertrag gewollt, so sollte insbesondere eine Abnahme ausdrücklich geregelt werden.

Eine klare Einordnung als Werk- oder Dienstvertrag fällt nicht immer leicht und hängt letztlich von der Ausgestaltung der Vertragsbeziehung im Einzelnen ab.

Es soll nicht unerwähnt bleiben, dass bei Open-Source-Software für die Nutzerinnen und Nutzer auch immer die Möglichkeit besteht, den Support selbst zu leisten, da durch Open-Source-Lizenzen alle nötigen Rechte eingeräumt werden, insbesondere auch das Recht zur Veränderung des Codes und der Weitergabe veränderten Codes. Wenn dazu nötige Kompetenz und Ressourcen vorhanden sind, kann dies eine Alternative zu eingekauften Support-Leistungen darstellen.

Support für Open-Source-Software ist oft gefragt genug, um daraus ein eigenständiges Geschäftsmodell zu machen. Ein Kriterium, um Support-Dienstleister zu bewerten, sind Beiträge zu den vom Support abgedeckten Open-Source-Projekten. Dabei ist jedoch zu beachten, dass dies nicht für jede Form der Dienstleistung das geeignete Kriterium ist. Für das Bereitstellen von Sichersheits-Updates ist es ein guter Indikator, für Beratungsleistungen nicht notwendigerweise. In jedem Fall gilt: Die Finanzierung eines Supportdienstleisters, welche die Software aktiv pflegt, wartet, monetär unterstützt oder Features beisteuert, stärkt auch das Projekt.

3.2.3 Long Term Support

Der »Long Term Support« (LTS) bietet Anwenderinnen und Anwendern eine besondere Art der Sicherheit: Die Produktversionen werden längerfristig mit wichtigen Bugfixes und Security-Upgrades versorgt. Dass das eine besondere Herausforderung werden kann, hat folgende Gründe:

Open-Source-Software wird von der Community in doppelter Hinsicht kontinuierlich weiterentwickelt. Zum einen geht es um die Bereitstellung neuer funktional angereicherter Versionen der Software. Zum anderen geht es um die Bereitstellung von Bugfixes und Security-Upgrades der funktional nicht erweiterten Version.

Anwenderinnen und Anwender, die nicht immer gleich auf die neuesten funktionalen Versionen umstellen wollen, haben also ein Interesse daran, dass die von ihnen verwendete Version über einen längeren Zeitraum mit Bugfixes und Security-Upgrades versehen wird, ohne dass damit ein funktionales Upgrade einhergeht. Distributorinnen und Distributoren von Open-Source-Software haben dafür das Konzept von »Long-Time-Support-Versionen« entwickelt. Hier sichert der Distributor für eine bestimmte Sammlung von Open-Source-Software zu, dass sie über einen längeren Zeitraum als üblich die erwünschten Bugfixes und Security-Upgrades ausliefert.
Oder anders gesagt: LTS-Versionen rücken zum Zweck der Risikominimierung die Softwarequalität in den Mittelpunkt, neue Features stehen zurück.

Stabile LTS-Versionen zeichnen sich in der Regel durch einen Feature-Freeze aus. Dabei wird der erreichte Softwarestand an einem bestimmten Punkt von der normalen Entwicklung abgezweigt und funktional eingefroren. Während in den Hauptentwicklungszweig des Source Codes neue Funktionen einfließen und mit neueren Versionen auch bereits ausgeliefert werden, werden für den abgezweigten Stand nur noch Bugs und Schwachstellen behoben. Die entsprechenden Patches können einzeln oder in neuen Releases (Maintenance oder Service Packs, Minor Releases etc.) zur Verfügung gestellt werden. Daher sind Updates seltener und bestehen – wenn überhaupt – aus Funktionen, die schon ausgiebig getestet wurden. Das soll die Gefahr minimieren, dass sich neue Bugs einschleichen oder bisherige Funktionen unabsichtlich beeinträchtigt beziehungsweise sogar außer Betrieb gesetzt werden. Ein typischer Zyklus für eine solche LTS-Version liegt bei zwei Jahren – kann aber auch kürzer oder länger sein oder relativ zu weiteren Releases ausfallen.

Die Variante des Feature-Freeze sollten Anbieter von Open-Source-Software in Überlegungen zu ihrer eigenen LTS-Strategie einfließen lassen, sowohl hinsichtlich der Dauer, für die alte Versionen noch unterstützt werden, als auch hinsichtlich der Veröffentlichung von speziellen LTS-Versionen. Ein weiteres Geschäftsmodell besteht mittlerweile darin, kommerziellen LTS über den durch die Community gewährleisteten Zeitraum hinaus anzubieten.

Nutzerinnen und Nutzer von Open-Source-Software sollten sich über die jeweiligen Regeln des LTS informieren. Fast alle Projekte haben eine eigene Policy, wie etwa die Eclipse Foundation oder die von der Firma Canonical erstellte Distribution Ubuntu. Für die langfristige IT- und Produktplanung sind die daraus zu gewinnenden Informationen von hohem Wert.[23]

23 Einen besonderen Fall stellen Hersteller von Customer Electronics dar, die Linux auf ihren Systemen zum Einsatz bringen: Hier wurde einer firmenübergreifende Long Term Support Initiative (LTSI) in Form eines industrieweiten Projektes ins Leben gerufen. Dieses Projekt legt Long Term Support für einen Industriezweig des Linux-Kernels fest, der als verlässliche Basis für Customer Electronics dient. (vgl. dazu etwa: https://ltsi. linuxfoundation.org/what-is-ltsi)

3.2.4 Schutz vor Ansprüchen Dritter

Beim Einsatz von Open-Source-Software kann es zur Verletzung von Rechten Dritter kommen.

Beim Einsatz von Open-Source-Software kann es zur Verletzung von Rechten Dritter kommen. Zu nennen sind hier insbesondere fremde Urheber- und Patentrechte, aber auch Markenrechte. Da Open-Source-Software im Hinblick auf solche Rechtspositionen vor ihrem Einsatz in der Praxis meist nicht näher überprüft und analysiert wird, besteht das nicht zu unterschätzende Risiko von Rechtsverletzungen und damit von teuren Abmahnungen, Unterlassungs- und Schadensersatzklagen sowie kostenintensiven Patentrechtsstreitigkeiten, initiiert durch die jeweiligen Rechteinhaberinnen und Rechteinhaber. Treffen kann es letztlich alle – diejenigen, die Open-Source-Software vertreiben, und diejenigen, die Open-Source-Software nutzen.

In ↗ Abschnitt 7.3.2 wird OpenChain und ISO 5230 behandelt. Hierbei handelt es sich um einen Industrie- und ISO Standard zur Reduktion der Open-Source-Compliance-Risiken im Unternehmen und der Lieferkette.

Der Lizenzgebende haftet im Fall kostenlos überlassener Software für Rechtsmängel nur eingeschränkt. Wurde weder vorsätzlich noch arglistig gehandelt – was oft der Fall sein dürfte –, bleiben die Anwendenden bzw. Distribuierenden auf dem Schaden sitzen; in diesem Fall tragen sie allein das Risiko von Mängeln.

Für diesen Fall bieten einige Distributoren einen speziellen, kostenpflichtigen Versicherungsschutz an. Solche Produkte, etwa unter dem Namen »Indemnification Program« oder »Assurance Program« vertrieben, räumen den Open-Source-Software-Anwendenden zusätzliche Ansprüche ein und sichern sie gegen die genannten Risiken finanziell ab. Solche Versicherungen decken in der Regel Schäden, die aufgrund von Patent- und Urheberrechtsverletzungen entstehen, in gewissem Umfang ab. In diesen Konstellationen sinkt das Risiko der Verletzung von Rechten Dritter auch dadurch, dass typischerweise mehr Aufwand bei den Distribuierenden in die Prüfung auf solche möglichen Rechte Dritter gesteckt wird.

Im Einzelfall sollte jedoch geprüft werden, ob diese Versicherungsprodukte auch die konkret beabsichtigte Nutzung der Software abdecken. So ist darauf zu achten, dass sich Ansprüche nicht lediglich auf den Fall der Eigennutzung beschränken, wenn ein Vertrieb der Software einzeln oder im Paket mit anderen Komponenten beabsichtigt ist. Auch deckt die Versicherung möglicherweise nur einen Kern der Distribution ab, nicht jedoch sämtliche, in der Distribution enthaltenen Programmteile.

3.3 Gütekriterien zu verwendender Open-Source-Software

Wer Open-Source-Software einsetzen möchte, sollte operative Aspekte und Gütekriterien jeder Software bei Open Source nicht außer Acht lassen. Dies gilt insbesondere, da Open-Source-Software bei Direktnutzung ohne SLAs oder vertraglicher Zusicherungen oder Haftung jeglicher Art kommt.

Besonders bei häufiger oder kritischer Verwendung von Frameworks, Libraries, Infrastruktur und Services wird schnell relevant, ob die Software etwa an ein Unternehmen geknüpft ist, wodurch Service und Support im zweifelsfall erwerblich sind, ob eine lebendige und diverse Community existiert und aushilft, welche möglicherweise finanzielle Bedarfe äußert, oder möglicherweise gar niemand oder nur wenige sich für die Software zuständig sehen.

Es gibt keine abschließende Kriterienliste, denn was zu einer Software relevant ist, hängt immer vom Einsatz ab. Häufig anzutreffen sind jedoch einige Aspekte, die im Folgenden zusammengefasst vorliegen.

- **Governance:** Wie werden Entscheidungen im Projekt getroffen und wer hat welchen Einfluss? Größere Projekte haben viele Beteiligte, die alle in irgendeiner Form Einfluss auf das Projekt ausüben. Wer führt das Projekt? Welche Regelungen zum geistigen Eigentum sind getroffen? Wer bezahlt Infrastruktur und/oder Wartung? Wie dies umgesetzt ist, muss in der Regel im Einzelfall geprüft werden. Einheitlich lässt sich aber sagen, dass die bewusste Unterstützung des Projektes durch Dritte, etwa durch eine etablierte Open-Source-Foundation, verschiedene rechtliche und organisatorische Aspekte oft bereits hinreichend adressiert.

- **Wartbarkeit und Support:** Open-Source-Software kann unentgeltlich eingesetzt werden – muss es aber nicht. Und oft ist nicht direkt erkennbar, ob für eine Software kommerzieller Support, wie in ↗ Kapitel 3.2.2 beschrieben, verfügbar ist. Je nach Einsatzzweck sollte eruiert werden, ob und welche Art des Supportes möglich ist.

- **Finanzielle und personelle Nachhaltigkeit:** Viele Open-Source-Projekte klagen über mangelnde Finanzierung oder wandelnde Interessenslagen der Entwickelnden. Oft ist der sog. Bus-Faktor auch deutlich geringer, als man vom Ruf einiger Projekte annehmen mag. Wer auf Open-Source-Software setzt, sollte sich die Frage stellen, ob eine Finanzierung dieses Ökosystems Mehrwert bietet. Häufig lässt sich hier das Angenehme mit dem Nützlichen verbinden, sofern Kontributoren der Software entgeltliche Service- und Support-Leistungen anbieten.

Wer auf Open-Source-Software setzt, sollte sich die Frage stellen, ob eine Finanzierung dieses Ökosystems Mehrwert bietet.

- **IT-Security:** Open Source ist keine Garantie für Sicherheit. Klar ist, dass durch die Wiederverwendung zum Beispiel spezialisierter kryptographischer Bibliotheken und die Möglichkeit externer Reviews zwar Qualität ermöglicht werden kann, dies aber keineswegs eine zwangsläufige Folge ist. Wer Open-Source-Software einsetzt, darf nicht vergessen, dass die Urheber möglicherweise keine Erfahrung im Umgang mit sicherheitsrelevanten Aspekten oder ausreichend Mittel für entsprechende Optimierungen haben. Software mit ausreichender Dokumentation, hohen Scores bei statischen und dynamischen Codeanalysen und transparent ausgewiesenen Mechanismen zur Qualitätssicherung kann oft deutlich leichter weiterentwickelt und gewartet werden und legt höhere Erfahrung der Entwicklerinnen und Entwickler nahe. Ein vertretbares oder unüberschaubares Bug/Issue-Backlog gibt zudem schnell weitere Aufschlüsse über die Stabilität.

- **Beisteuerbarkeit:** Open Source garantiert nicht, dass man sich beteiligen kann und auch wenn die meisten Open-Source-Projekte viele Kontributionen dankend annehmen, gibt es auch andere Interessen. Etwa für »Open Core«-Lösungen ist es unwirtschaftlich, wenn die Open-Source-Lösung zu gut beziehungsweise zu vollständig würde. Auch können Sprachbarrieren, unterschiedliche Qualitätsansprüche, CLAs und langfristige Vorstellungen der Projektentwicklung schnell zu einem Problem werden. Manchmal stehen sogar eigene Interessen einer Kontribution entgegen, etwa wenn die Kontribution Image oder Strategie widerspräche. Projekte weisen oft durch eine CONTRIBUTING-Datei aus, welche Richtlinien für die Kontribution gelten.

Verschiedene Projekte nähern sich diesem Problem aus unterschiedlichen Richtungen. Als beispielhafte Kriterienkataloge können dabei das ↗ Best Practices Program der Core Infrastructure Initiative oder das Open-Source-Tool ↗ fosstars genannt werden. Die aufgezeigten Herangehensweisen stellen eine unvollständige Liste von Möglichkeiten dar, Open-Source-Software zu bewerten. Es wird ausdrücklich empfohlen, mehrere Herangehensweisen in Kombination anzuwenden, um ein umfassenderes Bild in einem Auswahlprozess zu bilden.

Wichtig: Welche Open-Source-Software **wirklich** wichtig ist, fällt selten direkt ins Auge. Oft verwenden verschiedene Abteilungen unterschiedliche und spezialisierte Komponenten, kommunizieren aber unwissentlich über die gleichen kryptographischen Bibliotheken, basieren auf den gleichen Web-Frameworks oder nutzen die gleiche Datenbank. Welche Komponente nach diesem Gesichtspunkt somit wichtig oder gar unersetzlich ist, wird oft nur deutlich, wenn eine Abteilung, etwa ein OSPO, hierzu Statistiken pflegt.

3.4 Lizenzmanagement und Compliance

Unabhängig davon, welche der oben genannten Varianten gewählt wird: Wenn bereits existierende bzw. externe Open-Source-Software ein Teil der Lieferung oder Veröffentlichung darstellt, muss der Anbieter sicherstellen, dass die involvierten Lizenzen eingehalten werden. Grundlage hierfür bildet das Lizenzmanagement, dessen wesentliche Merkmale im Folgenden erläutert werden.

Das Lizenzmanagement umfasst folgende Aspekte:

1. Erfassung der verwendeten Lizenzen
2. Durchführung von Lizenzinterpretationen
3. Verwaltung von Lizenzinterpretationen
4. Definition der Umsetzungsmöglichkeiten von Lizenzinterpretationen
5. Verifikation und Erfassung der Umsetzung zur Lieferung.

Mehr zum Thema Compliance aus formaler Sicht findet sich in ↗Kapitel 7. Im Folgenden sollen einige praktische Aspekte näher beleuchtet werden.

3.4.1 Erfassung der verwendeten Lizenzen

Die systematische Erfassung der verwendeten externen Software und deren Lizenzen bildet die Basis aller weiteren Tätigkeiten und Maßnahmen des Lizenzmanagements.

Weder die systematische Erfassung von Third-Party-Software noch das Lizenzmanagement sind spezifisch für Open-Source-Software, sondern eine allgemeine Aufgabe, die jedes Unternehmen im Rahmen seiner Compliance-Tätigkeiten durchzuführen hat.

Alle Open-Source-Software-Komponenten, die auf Rechnern im Unternehmen installiert oder in Produkten des Unternehmens eingearbeitet sind, sollten in einem zentralen Register erfasst werden.

Das Ergebnis müsste kontinuierlich aktualisiert und gepflegt werden, damit auch Änderungen in der Konzeption systematisch berücksichtigt werden. Die mit Open-Source-Software befassten Mitarbeitenden müssten dazu die maßgeblichen Informationen – inklusive solche zur Bestimmung und Verwendung der Open-Source-Software im Unternehmen – immer vor Beginn einer Arbeit mit der Open-Source--Software eintragen.

Eine entsprechende firmeneigene Datenbank sollte folgende Informationen beinhalten:

- Name und Version der Open-Source-Software,
- Autorin bzw. Bezugsquelle der Open-Source-Software,
- Lizenzart und Version der Lizenz der Open-Source-Software,
- Beginn der Nutzung,
- Art der Nutzung:
  - Programm oder Bibliothek,
  - eingebettete Komponente oder eigenständige Einheit,
  - in modifizierter oder in unmodifizierter Form,
  - mit Weitergabe an Dritte oder ohne Weitergabe,
  - in Form von Binärdateien oder als Source Code,
- Name des mit der Nutzung der Open-Source-Software befassten Mitarbeitenden,
- Zielprojekt für den Einsatz der Open-Source-Software,
- geplante interne Nutzung der Open-Source-Software mit oder ohne Änderungen,
- geplante externe Verwendung (Kopie, Distribution) der Open-Source-Software mit oder ohne Veränderungen,
- Genehmigung des Open-Source-Software-Einsatzes durch die zuständigen Entscheidungstragenden.

Bei kommerzieller Third-Party-Software werden die Lizenzbedingungen zwischen Lizenzgebenden und Lizenznehmenden – wenigstens dem Prinzip nach – »verhandelt«. Bei Open-Source-Software sind die Lizenzen und ihre Bedingungen vorab generell bekannt und – de facto zumeist – nicht verhandelbar. So ist es praktisch eine Sache des Lizenznehmenden, also des Nutzenden von Open-Source-Software, die Lizenzen zu erfassen, die für ein komplettes (Open-Source-) Software-Paket relevant sind. Das bedeutet, dass im Ernstfall – also da, wo man sich nicht auf Vorarbeiten der Community verlassen kann – das ganze Paket nach Lizenzhinweisen durchsucht werden muss. Aufgrund der Komplexität und der Größe heutiger Softwareprojekte ist eine manuelle Suche praktisch nicht durchführbar und auch nicht sinnvoll. Eine Reihe von Softwarewerkzeugen steht zur Verfügung, welche die Identifikation der in einem Software-Paket zur Anwendung kommenden Lizenzen unterstützen. Zwei dieser Softwarewerkzeuge seien in diesem Zusammenhang besonders erwähnt, da diese selbst Open-Source-Projekte sind. Es handelt sich dabei um Open-Source-Review Toolkit[24] und FOSSology[25].

> Die meisten Open-Source-Lizenzen fordern, dass Nutzerinnen und Nutzer der Software die Lizenztexte mit der Software bereitgestellt bekommen.

Jedoch ist die alleinige Identifikation der involvierten Lizenzen und der Herkunft der jeweiligen Open-Source-Pakete nicht ausreichend für den Fall, dass das Open-Source-Paket entweder in binärer Form oder im Source Code weiterverteilt werden soll, zum Beispiel als Bestandteil eines Produktes oder einer Lösung. Die meisten Open-Source-Lizenzen fordern, dass Nutzerinnen und Nutzer der Software die Lizenztexte mit der Software bereitgestellt bekommen. Außerdem ist es zumindest gute Tradition, auch die Urheber der Software zu nennen. Die Art und Weise, wie und in welchem Format dies geschehen soll, ist oft nicht definiert und nur in manchen Lizenzen näher

24 vgl. Open-Source-Review Toolkit (ORT), https://oss-review-toolkit.org/
25 vgl. FOSSology: Advancing open Source analysis http://www.fossology.org/projects/fossology

beschrieben. Einzige Bedingung ist, dass diese Informationen für Menschen lesbar und an einem leicht zu findenden Ort hinterlegt sein müssen.

Das Fehlen eines standardisierten Formats für die Bereitstellung von Informationen für die Nutzerinnen der Software hat bisher dazu geführt, dass viele Unternehmen unterschiedliche, zueinander inkompatible Lösungen realisiert haben. Dies stellt besonders für Zuliefernde ein immer größer werdendes Problem dar, da diese möglicherweise die gleichen Informationen in vielen verschiedenen Ausprägungen und Formaten bereitstellen müssen. Das Problem wird seit geraumer Zeit durch die Linux Foundation im Rahmen des „Open Compliance Programs»[26] adressiert. Ein Element des Programms ist die Definition des SPDX Austauschformats[27], das die Auflistung der genutzten Komponenten, deren Dateien, sowie der involvierten Lizenzen und der entsprechenden Urheberinnen erlaubt. Dieser Standard ist in Version 2.2.1 als ISO Standard 5962:2021 verfügbar. Diese Vorgehensweise einer strukturierten Erfassung eingesetzter Open-Source-Komponenten ist unter dem Begriff Software Bill of Materials (SBOM) bekannt. Sie stellt zunehmend ein zentrales Mittel im Open-Source-Compliance-Prozess dar.

Letztlich mag sich die Open-Source-Eingangskontrolle bei den Unternehmen auf die Auswertung solcher Dokumente beschränken – vorausgesetzt, sie wurden von den Lieferantinnen mit entsprechender Sorgfalt oder unter Zuhilfenahme geeigneter Werkzeuge automatisiert erstellt. Gleichwohl ist es aber wichtig anzumerken, dass es die Pflicht des nutzenden Unternehmens ist, alle Bedingungen einer Open-Source-Lizenz auch tatsächlich zu erfüllen. Diese Pflicht auf die Zuliefernden zu verschieben ist nicht möglich.

Diese strukturierte und idealerweise standardisierte Erfassung eingesetzter Open-Source-Komponenten ist auch unter dem Begriff Software Bill of Materials (SBOM) bekannt. Sie stellt zunehmend ein zentrales Mittel im Open-Source-Compliance-Prozess dar.

3.4.2 Belastbarkeit der Lizenzinformationen verschiedener Open-Source-Ökosysteme

Ist die Rede von der Erfassung der Lizenzinformationen, wird davon ausgegangen, dass Open-Source-Projekte selbst ausreichend Lizenzinformationen ausweisen. Realität ist jedoch häufig, dass in verschiedenen Ökosystemen, insbesondere jenen, die schnelllebig oder komplex sind, oder die es für Open-Source-Einsteigende sehr einfach machen wollen, häufig nicht ausreichend Regeln für die Bereitstellung von Open-Source-Lizenzinformationen gelten.

In manchen Ökosystemen lässt sich etwa beobachten, dass Code häufig ohne Lizenzen veröffentlicht oder Lizenzinformationen falsch oder unvollständig ausgewiesen werden.

26 vgl. Linux Foundation: Open Compliance Program http://www.linuxfoundation.org/programs/legal/compliance
27 vgl. http://spdx.org – der Standard kommt auch in diesem Leitfaden zur Anwendung

Während Open-Source-Betriebssysteme häufig eher hohe Anforderungen an Compliance stellen und diese auch forcieren, zeigen andere Distributoren schon deutlich weniger Enforcement von Lizenzarten und Informationen. Am Ende gibt es Distributoren vollständiger Kompositionen wie Container-Images, welche in der Regel intransparent hinsichtlich der Lizenzen und Inhalte sind.

Wer Open Source verwendet, muss sich vergewissern, mit welchem Ökosystem er oder sie sich befasst, und mit welchen Zulieferungen er oder sie zu rechnen hat, oder welche Risiken bei der Datenlage zu erwarten sind. Häufig treffen hier der Wunsch nach Compliance und die Realität der Datenlage aufeinander. Etwa können theoretische Anforderungen in wenigen Ökosystemen trivial durchführbar sein und gleichzeitig in anderen zu unverhältnismäßigen Aufwendungen führen.

Dennoch gibt es verschiedene Lösungsmöglichkeiten. Wer etwa Open Source explorativ und im niederen Risikobereich einsetzt, kann oft einen wirtschaftlich angemessenen Mittelweg zwischen vollständiger und forensischer Compliance und akzeptiertem Risiko suchen. Zudem bietet sich oft ein Schulterschluss zwischen Open Source und Security an, denn oft sind die Lizenzrechtlich intransparentesten Lösungen in gleichem Maße intransparent für verschiedene Securitymaßnahmen. Ein Beispiel hierfür können Container-Images sein, welche nur nach expliziter Prüfung oder aus kontrolliertem internen Build für den Einsatz freigegeben werden.

3.4.3 Container-Compliance

Container stellen eine Methode zur Kapselung von Software dar, sowohl zur Laufzeit als auch zur Distribution. Manchmal werden Container als leichtgewichtige Form der Virtualisierung bezeichnet, tatsächlich kommt aber deutlich unterschiedliche Technologie zum Einsatz, die spezielle Herausforderungen für Compliance mit sich bringt.

Der zentrale Vorteil von Containern liegt in der Einfachheit der Handhabung. Mit wenigen Kommandos können Container Images heruntergeladen, weitere Container Images abgeleitet, ein Container instanziiert oder Container Images publiziert werden. Systeme werden aus einer Vielzahl von Containern aufgebaut, integriert und über spezialisierte Werkzeuge orchestriert.

> Der zentrale Vorteil von Containern liegt in der Einfachheit der Handhabung.

Aus dem einstigen Hype ist eine ernstzunehmende Technologie erwachsen, die ein wichtiges Fundament für zukünftige Innovation sein wird. Arbeitsabläufe, Test, Testbetrieb, Rollout und Produktivbetrieb werden erheblich vereinfacht, der massive Open-Source-Anteil ermöglicht eine hohe Transparenz und Anpassbarkeit. Zudem haben Container insbesondere im Vergleich zu Images von virtuellen Maschinen einen reduzierten Ressourcenbedarf.

Das Angebot Container-Technologie einzusetzen, erstreckt sich hierbei über die gesamten Nutzungsmöglichkeiten – von der rein internen Nutzung bis hin zu weltweiten

Auslieferungen von Lösungen mit unterschiedlichen unterstützenden Supportmaßnahmen, bis hin zu »sich selbst« aktualisierenden Komplettlösungen bereits in der Umsetzung.

Doch die Geschwindigkeit und Eleganz einer neuen Technologie war schon öfter ein trügerischer Schutz vor Versäumnissen und den damit einhergehenden Verstößen.

Die Einfachheit der Container-Technologie ermöglicht es, Software aus verschiedensten Quellen zusammen zu ziehen, auch in unstrukturierter Art und Weise. Jede Linux-Distribution handhabt Lizenz und Copyright-Informationen auffallend anders und auf sehr unterschiedlichen Qualitäts- oder Detailebenen. Die Patch-Policies sind grundlegend unterschiedlich, da sie verschiedenen Philosophien und Notwendigkeiten folgen. Zudem kann Software auch leicht direkt in Container integriert werden, ohne konsistent gebaute Pakete einer Linux-Distribution zu verwenden.

Die Einfachheit der Container-Technologie ermöglicht es, Software aus verschiedensten Quellen zusammen zu ziehen, auch in unstrukturierter Art und Weise.

Hierdurch entsteht ein großes Potential für Versäumnisse, sowohl in der eigenen Entwicklung als auch in der Entwicklung der Zuliefernden, also der gesamten Lieferkette. Denn während ein Container in seiner Außenwirkung die Komplexität eines Software-Stacks versteckt, gelten weiterhin für alle verwendeten Komponenten und deren Zusammenspiel die rechtlichen, regulatorischen und vertraglichen Verpflichtungen. Unbemerkt könnte diese neue Technologie bestehende Unternehmensrichtlinien unterwandern. Eine Möglichkeit, die nicht zu unterschätzen ist.

Aus dem Blickwinkel von License Compliance Managenden oder Spezialistinnen und Spezialisten für IT-Recht auf viele der aktuellen Containerergebnisse, liegt der Vergleich mit der Büchse der Pandora oft nah:

- Container-Inhalte werden oft aus einer Vielzahl von unterschiedlichen Quellen bezogen, bei denen sich nicht ohne weiteres Vertrauenswürdigkeit von Quellen und Inhalten sicherstellen lassen.
- Oftmals sind Lizenzhinweise unscharf, unvollständig oder wurden sogar vollständig entfernt (Stichwort: Pico-Container).
- Die Frage nach der Plausibilität, Vereinbarkeit und Erfüllung der Lizenzverpflichtungen bleibt bereits an der Quelle eines Basis Container Images unbeantwortet.
- Ohne weitere Informationen, Werkzeuge und Aufwand kommt das Unternehmen kaum zu einer Aussage, wie sicher die Software im Container ist und welche bekannten Sicherheitslücken von Relevanz sein können.
- Die Technologie geht über den Rahmen eines Containers hinaus. Das heißt: Es gilt, die Lizenz jedes Container Images und der darin enthaltenen Inhalte zu überblicken.

Ohne die geeigneten Unterstützungswerkzeuge und einen Dialog der Disziplinen im Unternehmen ist eine realistische Bewertung kaum möglich. Zudem jedes Unternehmen auch gegenüber diesen neuen Technologien dringend zu einem eigenen Rechtsverständnis und einer eigenen Bewertung und Positionierung kommen sollte.

Eine einseitige ad-hoc Bewertung, eine unmittelbare Nutzung oder Weitergabe erscheinen in diesem Licht unprofessionell und fahrlässig.

Die Situation ist prinzipiell nicht aussichtslos. Die grundsätzlichen Vorgehensweisen zu Open-Source-Compliance lassen sich auch auf Container anwenden. Allerdings stellen sich durch die Eigenarten der Container-Technologie spezielle Herausforderungen, zum Beispiel:

- Die Konstruktion von Containern aus verschiedenen sogenannten Layern erfordert, das prinzipiell für jedes Layer jeweils Lizenz-Compliance hergestellt werden müsste
- Es fehlen standardisierte Verfahren zur Distribution von zugehörigem Quellcode, im Gegensatz zu beispielsweise Linux-Distributionen, die dafür ausgereifte Verfahren bieten
- Signatur-Verfahren von Containern zur Sicherstellung der Integrität der Inhalte ist noch nicht sehr ausgereift

> Um zukunftssicher, handlungsfähig, konkurrenzfähig und schuldlos zu bleiben, muss Container-Compliance mit hoher Aufmerksamkeit betrachtet werden.

Eine einheitliche Haltung der entwickelnden Industrie in Deutschland und international ist in diesem Zusammenhang momentan noch nicht erkennbar. Es gibt zwar eine Reihe von Werkzeugen, die aber noch nicht alle Fragen der Compliance abdecken können. Um zukunftssicher, handlungsfähig, konkurrenzfähig und schuldlos zu bleiben, muss Container-Compliance mit hoher Aufmerksamkeit betrachtet werden.

Sieht ein Unternehmen vor, Container-basierte Lösungen in einem produktiven Betrieb zu nutzen oder an weitere Empfängerinnen und Empfänger in einer Lieferkette weiterzugeben, sind die Umsetzungen der nachfolgenden Handlungsempfehlungen wünschenswert:

- Externe Container Images werden nur aus ausgewählten Quellen bezogen.
- Externe Container Images sind entsprechend eigenen Richtlinien zu verifizieren.
- Ableitungen von eigenen Container Images können über definierte Abläufe reproduziert werden.
- Modifikationen an Fremdkomponenten sind durch eigene Richtlinien eingeschränkt und werden durch Reviews oder Werkzeuge beim Erzeugen der Ableitungen geprüft.
- Aktive Inventarisierung der Software-Komponenten – insbesondere der Komponenten Dritter – zur Prüfung gegen eigene Richtlinien.

Dabei sind die Inhalte der aufgeführten eigenen Richtlinien hier offengelassen. Letztendlich sind die Ziele dieser Richtlinie Sache des Unternehmens beziehungsweise ergeben sich aus den regulatorischen, gesetzlichen und vertraglichen Rahmenbedingungen aus der Perspektive und dem Risikoappetit der Unternehmensführung.

Von einer Microservice Architektur basierend auf einer Serverless Infrastructure, die das Prädikat Compliance-by-Design trägt und dem wachsenden gesellschaftlichen Transparenzverständnis gerecht wird, sind wir – mit den aktuell heterogenen Inhalten, existierenden Standards und Qualitätsargumenten – in der Umsetzung wohl noch weit entfernt.

3.4.4 Durchführung und Verwaltung von Lizenzinterpretationen

Auf die systematische Erfassung der zur Anwendung kommenden Lizenzen folgt die Interpretation der in den Lizenzen eingeräumten Rechte und auferlegten Pflichten sowie die Verwaltung der Interpretation. Ziel der Interpretation der Lizenzen ist es, Urheberrechtsverletzungen zu vermeiden sowie Klarheit darüber zu bekommen, ob die Lizenzen mit den eigenen Absichten vereinbar sind. Rechtliche Expertise ist bei dieser Tätigkeit unerlässlich. Die Interpretation der Lizenzen ist genauso wie deren Erfassung systematisch durchzuführen und wiederverwendbar zu speichern, um zufällige Fehler auszuschließen. Welche Technologie zur Verwaltung der Interpretationen verwendet wird, ist frei wählbar. Eine Datenbank-orientierte Lösung ist aber sinnvoll – besonders dann, wenn der Prozess der Erfassung und Ausleitung der zu erfüllenden Lizenzbedingungen aus Gründen der Effizienz weitestgehend automatisiert werden soll.

Bestehende Lizenzinterpretationen sind zu pflegen und gegebenenfalls an sich neu ergebende Aspekte der Rechtsprechung anzupassen.

3.4.5 Möglichkeiten zur Umsetzung von Lizenzinterpretationen

Der Interpretation der Lizenzbedingungen folgt die Definition der Umsetzung der Interpretationen. Die Umsetzung ist in der Regel spezifisch für ein konkretes Lieferszenario:

In diesem Schritt müssen die Spezifika der einzelnen Produkte und Lösungen beachtet werden, in denen die erfassten Open-Source-Pakete ganz oder teilweise mit oder ohne Modifikationen enthalten sind. So kann zum Beispiel die Verpflichtung, den Lizenztext der Software beizulegen, im Falle einer vorhandenen GUI (Graphical User Interface) einfach durch einen Button »show license information« im Hauptmenü umgesetzt werden. Ist kein GUI vorhanden, können die Lizenztexte zum Beispiel auf der Produkt-CD oder auf einer dem Produkt beiliegenden CD als Textdateien bereitgestellt werden. Es ist sinnvoll, bei der konkreten Umsetzung »Best Practices« zu definieren und sich so weit wie möglich bei der Umsetzung der Lizenzinterpretationen an diesen auszurichten. Das minimiert den Aufwand durch eine Art Standardisierung und das Risiko, eine Lizenzinterpretation in einem konkreten Produkt falsch umzusetzen. Außerdem wird ein unternehmensweites, einheitliches Konzept im Laufe der Zeit als »Brand« wahrgenommen. Als Beispiel zum letzten Punkt: Allen Produkten des Unternehmens A liegt eine CD mit Namen »Open-Source-Software« bei, auf der alle Angaben und der entsprechende Source Code der gelieferten Open-Source-Software-Pakete enthalten sind.

Es ist sinnvoll, bei der konkreten Umsetzung »Best Practices« zu definieren und sich so weit wie möglich bei der Umsetzung der Lizenzinterpretationen an diesen auszurichten.

Ein effizientes Verfahren, um die Einhaltung der Lizenzbedingungen für konkrete Produkte und Lösungen sicherzustellen, ist die Behandlung der Lizenzbedingungen

als Requirements, welche die einzelnen Lieferungen erfüllen müssen. Dazu wird beispielsweise die Forderung einer Lizenz, den Lizenztext jeder Lieferung der Software beizulegen, als »mandatory requirement« des Produktes in das verwendete Requirements Engineering Werkzeug eingepflegt. Die konkrete Art der Umsetzung für das jeweilige Produkt wird in diesem Werkzeug dokumentiert und verfolgt. Das Höchstmaß an Effizienz und Vollständigkeit wird dann erreicht, wenn die Lizenzbedingungen (also das Ergebnis der Tätigkeit »Durchführung von Lizenzinterpretationen«) wie weiter oben empfohlen in einer Datenbank zusammen mit den definierten Best Practices hinterlegt sind. Diese werden entsprechend der in der jeweiligen Softwarelieferung enthaltenen Open-Source-Software automatisch aus der Datenbank ausgelesen und in das Requirements Engineering Werkzeug eingepflegt. Durch diese Art der Automatisierung wird die Vollständigkeit der »License compliance requirements« pro Produkt oder Lösung sichergestellt und auch deren konkrete Umsetzung ist verfolg- und überprüfbar dokumentiert. Dies wiederum vereinfacht die letzte Tätigkeit »Verifikation der Umsetzungsmöglichkeiten« deutlich.

3.4.6 Verifikation und Erfassung der Umsetzung zur Lieferung

Die Umsetzung der zu erfüllenden Lizenzbedingungen zu verifizieren und zu erfassen, ist aus quantitativen Gründen fast nur noch mit Werkzeugunterstützung möglich. Selbst in kleinen DSL-Routern oder DVD-Playern kommen teilweise ganze Linux-Distributionen mit Dutzenden von Open-Source-Paketen zum Einsatz. In manchen Sprach-Ökosystemen kann die Anzahl verwendeter Module auch mehrere Tausend betragen. Eine Überprüfung, ob alle Lizenzbedingungen eingehalten werden, ist alleine aufgrund dieser schieren Menge ohne Softwareunterstützung nicht mehr möglich. Wie oben bereits beschrieben, sind alle Tätigkeiten des Lizenzmanagements so weit wie möglich zu automatisieren. Ein Hantieren mit Listen und Tabellen, in die Ergebnisse, Interpretationen und Umsetzungen manuell eingetragen und gepflegt werden müssen, wird selbst bei kleinen Lieferungen zwangsläufig zu Fehlern führen.

Lizenzmanagement ist eine den Einsatz von Software flankierende Tätigkeit, egal um welchen Typ Software es sich handelt – sei es Open-Source-Software oder proprietäre Software. Wichtig ist, dass das Lizenz-Management für jedes Paket und jedes eingesetzte Release der Pakete durchgeführt wird. Denn die Art und Menge der in einem Open-Source-Software-Paket zur Anwendung kommenden Lizenzen kann in jedem Release unterschiedlich zum Vorgänger sein. Auch auf sich ändernde Lizenzversionen ist zu achten, da damit neue Lizenzpflichten einhergehen können ebenso wie neue Rechte.

> Lizenzmanagement ist eine den Einsatz von Software flankierende Tätigkeit, egal um welchen Typ Software es sich handelt – sei es Open-Source-Software oder proprietäre Software.

4 Erstellen von Open-Source-Software

Aus Open-Source-Lizenzen ergibt sich keine Verpflichtung, an Open-Source-Software mitzuarbeiten, und je nach Lizenz ist es nicht einmal erforderlich, Änderungen zu veröffentlichen. Dennoch wird eine intensive Nutzung von Open-Source-Software fast zwangsläufig zum Thema Kontribution führen, da es eine Reihe von sehr guten wirtschaftlichen und technischen Gründen gibt, sich aktiv an Open Source zu beteiligen:

- Die Interaktion mit der Open-Source-Community und der direkte Kontakt zu Open-Source-Entwicklerinnen und -entwicklern stellt eine hervorragende Gelegenheit für Feedback und Lernen dar. Dies führt zu höherer Software-Qualität und verbesserter Fähigkeit, Support für verwendete Open-Source-Software zu leisten.

- Mittelfristig verringert eine Kontribution eigener Änderungen die Wartungskosten, da nach der Kontribution die Änderung Bestandteil der Open-Source-Software wird und somit in nachfolgenden Versionen automatisch enthalten ist. Ohne Kontribution muss die Änderung bei jedem Update mühsam nachgezogen und gegebenenfalls angepasst werden.

- Einfluss auf Open-Source-Projekte und deren Mitgestaltung kann in der Regel nur durch aktives Mitarbeiten und Beitragen von Code erreicht werden.

- Die Nachhaltigkeit von genutzter Open-Source-Software hängt entscheidend davon ab, dass genügend Nutzer sich auch an der Pflege und Weiterentwicklung beteiligen. Hier kann durch Beteiligung gezielt die Belastbarkeit von strategisch genutzter Software verbessert werden.

- Vielfach wird auch eine moralische Verpflichtung gesehen, nicht nur zu nehmen, sondern auch zu geben. Organisationen, die dieser Verantwortung nachkommen und Open-Source-Software nicht nur nutzen, sondern auch dazu beitragen, spüren dies in gesteigerter Reputation unter anderem als Arbeitgebende für die umworbenen Talente.

- Durch strategisches Engagement in und mit Open-Source-Projekten können Standards etabliert und Märkte gestaltet werden.

- Open-Source-Modelle können eine wirksame Methode sein, Zusammenarbeit und Kontakt mit Kunden, Partnern und Zuliefernden zu verbessern.

- Da Open-Source-Lizenzen diskriminierungsfreien Zugang zu Software garantieren, kann dies als Grundlage für konsortiale Software-Entwicklung dienen, ohne dass der Markt behindert oder einseitig beeinflusst wird und damit kartellrechtliche oder andere regulatorische Probleme entstehen

- Für eine zunehmende Zahl an Software-Entwicklerinnen und -Entwicklern kann die Möglichkeit, sich an Open-Source-Projekten zu beteiligen, ein starkes Argument sein, das die Attraktivität eines Arbeitgebenden bestimmt.

> Die Nachhaltigkeit von genutzter Open-Source-Software hängt entscheidend davon ab, dass genügend Nutzer sich auch an der Pflege und Weiterentwicklung beteiligen.

Ein höheres Engagement bei Kontributionen bis hin zum Eröffnen von neuen Open-Source-Projekten entsteht dann, wenn diese existierende oder neue Business-Modelle unterstützen sollen. Hierzu sei auf das ↗ Kapitel 5 verwiesen, in dem unterschiedliche Archetypen von Business-Modellen vorgestellt werden.

Das Thema Open-Source-Kontributionen bildet also ein weites Feld, in dem es einige Aspekte zu beachten gibt. Im Folgenden wollen wir auf das Thema Open-Source-Projekt-Governance, also die Managementprozesse rund um ein Open-Source-Projekt sowie weitere Grundlagen, wie zum Beispiel Projekttypen; die verschiedene Möglichkeiten, zu einem Open-Source-Projekt beizutragen sowie typisches Tooling rund um die Kommunikation und Entwicklung in Open-Source-Projekten eingehen.

4.1 Open-Source-Governance

Open-Source-Projekte sind in der Regel keine unbeschränkten Source Code Repositories, in denen jede nach Belieben Software ändern kann. Eine Open-Source-Lizenz sichert prinzipiell zu, die Software frei zu nutzen und an eigene Besonderheiten anzupassen. Sie definiert jedoch kein Anrecht auf die Übernahme eigener Änderungen in das ursprüngliche Projekt. Für die Weiterentwicklung von Open-Source-Projekten kommen daher häufig recht strenge Review- und Entscheidungsprozesse zum Einsatz, mittels derer die Qualität der Software dauerhaft erhalten wird. Diese Prozesse sowie die generellen Entscheidungs- und Steuerungsmechanismen eines Open-Source-Projektes, werden unter dem Begriff »Open-Source-Governance« zusammengefasst.

4.1.1 Kontributions-Pyramide

Die Community eines Open-Source-Projektes stellt typischerweise eine Pyramide dar. Fundament dieser Pyramide sind die Nutzerinnen und Nutzer der Software, im Speziellen die engagierten, die sich aktiv an der Community beteiligen, beispielsweise in Form von Bug Reports und Featurewünschen oder durch Beiträge in Mailinglisten. In der Pyramide direkt darüber sind Kontributorinnen und Kontributoren angesiedelt. Dies sind Teile der Community, die eigene Codebeiträge als Kontributionen vorschlagen. Sie haben in der Regel keine Schreibrechte auf das Repository. Ihre Beiträge werden von Maintainern des Projekts begutachtet und nach Erreichen der projekt typischen Qualität in das Repository übernommen.

An der Spitze der Pyramide stehen die Maintainer oder Committer eines Projekts. In dieser Rolle hat eine Entwicklerin eine höhere Verantwortung. Dies äußert sich zum Beispiel darin, dass sie Kontributionen annehmen kann. Dieses Recht äußert sich häufig durch Schreibrechte auf das Repository. Auf dieser Ebene findet die Kontrolle über die Software, deren Qualität und Funktionsvielfalt statt. In komplexeren Projekten kann diese Stufe noch weiter aufgefächert werden. Bekannt ist zum Beispiel der Linuxkernel, in dem es Subsystemmaintainer auf mehreren Stufen gibt und letztlich mit Linus Torvalds nur ein einzelner Entwickler die Patches in das Projektrepository übernimmt. Ein weiteres Beispiel sind Projekte der Eclipse Foundation, in diesen gibt es typischerweise noch einen Projekt-Lead, die zusätzliche Rechte im Kontext des Eclipse Foundation Entwicklungsprozesses hat, so kann sie zum Beispiel den Releaseprozess auslösen oder die formelle Wahl von Committern oder Projekt-Leads initiieren.

4.1.2 Projekte mit informeller Governance

So eine Kontributions-Pyramide kann chaotisch oder selbstorganisiert entstehen. Viele Open-Source-Projekte werden von einer Person als Erfinderin gestartet. Jedenfalls im ersten Wachstum wird so eine Person die entscheidende Rolle an der Spitze des Projekts behalten. Bei manchen Projekten bleibt die Person als wohlwollende Diktatorin auf Lebenszeit in dieser Rolle bestehen.

Bei langlebigen und wachsenden Projekten finden aber auch häufig Generationswechsel statt. Hier kommen dann neue Personen an die Spitze die sich in erster Linie durch ihr Engagement und den Umfang ihrer Kontribution auszeichnen.

Solche Projekte kommen meist sehr gut und lange ohne formelle Governance aus. Als konstituierendes Element reicht ein öffentliches Source Code Repository und eine Mailingliste.

Bezüglich der Beteiligung stellen diese Projekte aus einer Firmenperspektive sicherlich die größte Herausforderung dar. Um überhaupt Akzeptanz und später möglicherweise Einfluss auf so ein Projekt zu gewinnen, müssen zunächst Kompetenz und Bereitschaft für eine dauerhafte Bindung an das Projekt praktisch unter Beweis gestellt werden. Weil das bei informell organisierten Projekten immer von einzelnen Personen abhängt, birgt das unternehmerische Risiken.

Durch die fehlende Rechtseinheit und informelle Governance ist außerdem eine rechtliche Bewertung eines solchen Projekts schwierig und somit die Risikobewertung eines Beitrages unsicher. Beispielsweise stellen sich potentiell kartellrechtliche Fragen, wenn in einem ungeregelten Umfeld mit anderen Firmen kollaboriert werden soll.

Schließlich fehlt bei Projekten diesen Typs naturgemäß die formelle Regelung für Konflikt und Krisenfälle. Das muss kein Problem sein, es kann sich aber unter Umständen negativ auf die Langlebigkeit und Dynamik eines Projektes auswirken.

4.1.3 Charter-basierte Open-Source-Projekte

Gerade im industriellen Kontext gibt es oft das Bedürfnis nach formellerer Governance. Zum Beispiel bei vertikalen Ökosystemen gibt eine Reihe von Projekten und Projektfamilien, die eine Form von Mitgliedschaft mittels einer Charter definieren. Die Idee dieses Ansatzes ist es, die Bindung der an dem Projekt interessierten Organisationen sicherzustellen, indem die Mitgliedschaft mit einem jährlichen Beitrag oder einer Verpflichtung zur Bereitstellung von EntwicklungsresSourcen gekoppelt wird. Somit hat das Projekt ein Basisbudget, mit dem die Entwicklung vorangetrieben werden kann.

Die Mitgliedschaft geht dabei typischerweise mit Kontrollrechten einher. Steuergremien bestimmen über die Nutzung des Projektbudgets und sie definieren Arbeitsgruppen, in denen konkrete Themen vorangetrieben werden. Mitglieder haben dabei ein Recht auf Beteiligung an den unterschiedlichen Gremien und somit eine stärkere Kontrolle über das Projekt.

Die Projekte sind aber auch in diesem Fall Open-Source-Projekte, das heißt alle haben prinzipiell das Recht auf Beteiligung in Form von Kontributionen und können das Projekt entsprechend der gewählten Lizenz frei nutzen. Häufig eröffnen die Projekt-Charter auch explizit Möglichkeiten, wie frau durch aktive Teilnahme am Projekt weitere Rechte und Einflussmöglichkeiten erhalten kann. Generell stellt sich aber bei Kontribution zu Projekten diesen Typs die strategische Frage, ob ein möglicherweise gewünschter Einfluss das Eingehen einer Mitgliedschaft in dem Projekt notwendig macht. Erst wenn die geplanten Kontributionen einfache Bugfixes und minimale Patches übersteigen, ist eine Mitgliedschaft schnell angeraten.

4.1.4 Foundation-basierte Open-Source-Projekte

Aus einer Reihe erfolgreicher Projekte haben sich aus dem vorherigen Modell einer Kontrolle durch eine Non-Profit Unternehmung einzelne Foundations herausgebildet, die ihren Scope über die Jahre vergrößert haben. Beispiele hierfür sind die Linux-, die Eclipse- oder die Apache-Foundation. Diese vereinen eine Vielzahl von Open-Source-Projekten aus unterschiedlichen Bereichen. Auch definieren sie häufig einen standardisierten Governance-Rahmen, der etabliert ist und das Erstellen eines neuen Projekts erleichtert. Statt wie im vorherigen Beispiel eine komplett neue Governance, zum Beispiel in Form eines Vereins, selbst aufzuziehen, können in den Foundations etablierte Prozesse genutzt und durch den Support der Foundation für die Prozesse komplette Governance-Anteile ausgelagert werden.

Für eine kontributierende Firma ist dieses Konstrukt sicherlich das komfortabelste. Die Governance-Struktur ist standardisiert und eine Kontribution in mehrere Projekte einer Foundation erfordert nur beim ersten Projekt eine umfassende rechtliche Prüfung. Die Qualitätsstandards der Foundations sorgen für eine hohe Prozess- wie auch Softwarequalität, so dass eine langfristige Bindung an ein Open-Source-Projekt typischerweise geringe Risiken mit sich bringt.

> Eine langfristige Bindung an ein Open-Source-Projekt bringt typischerweise geringe Risiken mit sich.

4.1.5 Offenheit von Governance-Modellen

Ein Unternehmen oder ein Hersteller, der sich dazu entscheidet ein Softwareprojekt unter eine Open-Source-Lizenz zu stellen gibt damit offenbar einen wesentlichen Teil der Kontrolle über sein Produkt auf.

Auch wenn wir sehen mit welcher Dynamik sich die existierenden Open-Source-Projekte entwickeln und in welchem Umfang Innovation als Open-Source-Software neu entsteht, wird dieser Kontrollverlust
aus unternehmerischer Sicht vielleicht als besonderes Risiko wahrgenommen. Da liegt es nahe, durch die Konstruktion eines Governance Modells die Kontrolle über das Open-Source-Projekt zu sichern.

Anders als die Open-Source-Lizenzen, die einer Open-Source-Definition folgen, die durch die OSI geprüft und anerkannt ist, gibt es für die Governance Modelle noch keine festen Regeln.

Deshalb sollte vor einer Beteiligung an einem existierenden Projekt gegebenenfalls geprüft werden, ob die Governance-Regeln tatsächlich und ausreichend den Charakter einer Community-orientierten gemeinsamen Entwicklung und Zusammenarbeit ermöglichen. Im Interesse des Entstehen eines starken Community-Prozesses gilt das gleichermaßen für die Formulierung eigener Regeln bei der Gründung eines neuen Projektes.

Unter dem Schlagwort »Open Governance«[28,29] lassen sich Kriterien identifizieren, die ein Projekt erfüllen muss, um die Offenheit, die für den Code durch die Lizenz bestimmt ist, auch im Governance Modell abbildet. Insbesondere ist es offen für neue Teilnehmer und es ist auf Kooperation und den kreativen Prozess ausgerichtet.

Bei einem offenen Governance-Modell geht es um Fragen wie beispielsweise

- Verteilter oder neutraler Besitz der Nutzungsrechte von kontributiertem Code
- Neutraler Besitz der Infrastruktur, wie zum Beispiel die Internet-Domäne des Open-Source-Projekts, aber auch Aspekte wie von Namens- und Bildrechten an Projektressourcen
- Lizenzierung des Projektbrandings für zum Beispiel proprietäre Produkte
- Transparenz bezüglich
  - Entscheidungsprozesse
  - Code-of-Conduct und Prozess zur Einhaltung
  - Management des Projektbudgets, sollte ein solches erhoben werden (siehe charter-basierte Open-Source-Projekte)
  - Prozess zur Ernennung/Retirement von Maintainern/Committern
  - Roadmap-Prozess
  - Releasemanagement

> Eine offene Projekt-Governance ist ein Qualitätsmerkmal.

Eine offene Projekt-Governance ist ein Qualitätsmerkmal, das sicherstellt, dass einzelne Communityteilnehmerinnen und -teilnehmer einen klar definierten Einfluss auf das Projekt haben und die Prozesse im Projekt für alle beteiligten transparent sind. Eine übermäßige Einflussnahme soll somit vermieden werden.

28 Meijer, Lips, Chen: Open Governance https://www.frontiersin.org/articles/10.3389/frsc.2019.00003
29 Siehe zum Beispiel Open-Governance-Modelle im Kontext der Cloud Native Computing Foundation (CNCF): https://opengovernance.dev/

4.1.6 Spaltung von Open-Source-Projekten

Die Open-Source-Lizenzen garantieren allen Nutzerinnen und Nutzern und damit auch allen Entwicklerinnen und Entwicklern die Freiheit, eigene Versionen der Software anzubieten und zu verbreiten. In dem Fall, dass eine größere Contribution von der oben angesprochenen Führungsstruktur eines bestehenden Projektes nicht akzeptiert wird, bedeutet das, die veränderte Software kann in einem abgespaltenen Projekt unabhängig weitergeführt werden.

Die Spaltung einer Community kann Innovation beleben und neue Dynamik in ein möglicherweise schwerfälliges Projekt bringen.

Dieser Schritt will aber gut überlegt sein. Die Spaltung einer Community kann Innovation beleben und neue Dynamik in ein möglicherweise schwerfälliges Projekt bringen. Das war zum Beispiel bei der Abspaltung der 386er-Community von Andrew Tanenbaum's Minix-Projekt der Fall. Daraus ist der Linux Kernel entstanden.

Andererseits bedeutet so eine Spaltung aber möglicherweise auch die Abkopplung von Integration und Sicherheit im Ursprungsprojekt. Die Rückportierung und Vereinigung solcher Patches aus dem Ursprungsprojekt sind bis zu einem gewissen Grad möglich, sie bedeuten aber einen erheblichen Mehraufwand. Auch bringt jede Spaltung nicht unerhebliche Verunsicherung und Reibungsverluste.

4.1.7 Umgang mit IP- und Urheberrechten

Ein weiterer Aspekt der Projekt-Governance ist der Umgang mit IP- und Urheberrechten der Kontributierenden. Die IP-Rechte, zum Beispiel vom Source Code berührte Patente, sind in der Regel durch die genutzte Open-Source-Lizenz geregelt. Einige Lizenzen haben spezielle Patentklauseln die in der Regel ein unentgeltliches Recht zur Nutzung der involvierten Patente einräumen. Dies ist bei der Entscheidung zur Kontribution in ein Projekt ein relevanter Aspekt.

Betrifft eine Kontribution eigene Patente, so ist es in der Regel erstrebenswert, ein Projekt zu nutzen, welches eine Lizenz mit Patentklausel verwendet. Eine ungeklärte Patentsituation erhöht das Risiko bei Nutzung der Software und verringert somit die Attraktivität des Projekts.

Eine ungeklärte Patentsituation erhöht das Risiko bei Nutzung der Software und verringert somit die Attraktivität des Projekts.

Formell kann die uneingeschränkte Nutzung über ein Contributor Licence Agreement (CLA) oder als tatsächliche Übertragung des Copyright an die Entität durch ein Copyright Transfer Agreement geregelt werden. Für die konkrete Gestaltung eines CLA gibt es viele Beispiele[30], in denen unter anderem auch die Re-Lizenzierung als proprietäres Produkt geregelt werden kann.

30 https://en.wikipedia.org/wiki/Contributor_License_Agreement

4.1.7.1 Zuordnung des »Copyrights«: Individuum oder Entität

Mindestens bei den Projekten mit informeller Governance behält die Kontributorin alle Urheberrechte. Der Code setzt sich demnach aus Bestandteilen mit verschiedenen Eigentümerinnen und Eigentümern zusammen, der durch die gemeinsame Open-Source-Lizenz für andere nutzbar gemacht wird. Ein Vorteil dieser Vorgehensweise ist eine geringere Einstiegshürde, da hier im besten Fall neben der Open-Source-Lizenz keine weitere Regelung getroffen werden muss. Somit ist die Prüfung einer möglichen Kontribution einfacher und weniger eingeschränkt. Außerdem wird durch das Spektrum an Inhaberinnen und Eigentümern die Unabhängigkeit des Projekts gefördert, es wird schwerer für Einzelne, das Projekt zu dominieren. Die Nachteile dieser Vorgehensweise sind bereits oben im Abschnitt über informelle Governance beschrieben. Insbesondere lässt sich die Lizenz kaum mehr verändern. Selbst das Springen auf eine neuere Version einer Open-Source-Lizenz ist, sofern nicht frühzeitig bei Erstellung explizit geregelt, nur mit Einverständnis aller Copyrightinhaberinnen und -inhabern möglich. Das ist bei einem erfolgreichen Projekt nach ein paar Jahren quasi unmöglich.

In einem verfassten Governance Modell kann im Unterschied dazu vereinbart werden, dass eine Kontributorin die im Urheberrecht definierten Nutzungsrechte an eine Entität hinter dem Projekt zu deren freien Nutzung überträgt (unrestricted republishing rights). So eine Entität kann eine einzelne Firma oder ein rechtliches Konstrukt, wie zum Beispiel eine Foundation oder ein Verein, sein. Hier findet also eine Trennung statt zwischen der Lizenz auf eingehende Kontribution (Inbound) und der Lizenz der fertigen Software (Outbound). Der Vorteil hier besteht darin, dass das Copyright einer einzigen starken Entität (wie beispielsweise der Free Software Foundation) gehört, die es gegebenenfalls gegen Verletzungen und Angriffe verteidigen kann. Auch grundlegende Änderungen, wie zum Beispiel ein Update der Lizenz auf eine neuere Version, stellen damit kein Problem dar. Ein Nachteil dieses Verfahrens kann sich daraus ergeben, dass es eine dominierende Fraktion gibt, die über das Projekt und die Beiträge der Kontributorin bestimmen kann. Dies stellt gerade bei firmen-dominierten Projekten ein Risiko dar. Eine bisher freie Software kann hier auch in ein proprietäres Produkt gewandelt werden.

4.1.7.2 Schutz vor fremdem »Copyright«

Ein Open-Source-Projekt gibt die Nutzung der enthaltenen Software unter der Open-Source-Lizenz frei. Eine Nutzerin muss sich daher darauf verlassen können, dass alle Copyrightinhaberinnen und -inhabern des Projekts auch der Nutzung ihres Copyrights unter der Lizenz zugestimmt haben. Ein fälschlicherweise eingebrachter Inhalt von einer dritten Copyright-Inhaberin kann dazu führen, dass zumindest die Teile, die dieses Copyright enthalten, nicht nutzbar sind. Daher ist es im Interesse des Open-Source-Projekts, dafür zu sorgen, dass jede Kontributorin nur Inhalt einbringt, dessen Copyright sie besitzt oder für den sie eine entsprechende Lizenz hat. Das kann zum Beispiel Inhalt eines anderen Open-Source-Projekts sein, das unter einer kompatiblen Lizenz freigegeben wurde.

Zur maximalen Sicherung dieser Aspekte bedienen sich manche Open-Source-Projekte spezieller Vereinbarungen wie dem oben erwähnten Contributor License Agreement (CLA). Alternativ gibt es leichtgewichtige Mechanismen wie einem Developer Certificate of Origin (DCO), in dem die Autorin der Kontribution lediglich das Eigentum oder das Recht auf Weitergabe des Materials erklärt. Im einfachsten Fall gilt die »Inbound = Outbound«-Regel, die letztlich basierend auf der gewählten Open-Source-Lizenz Kontributionen akzeptiert, eine weitere Regelung findet nicht statt.

4.2 Wie können sich Unternehmen an Open-Source-Projekten beteiligen/Contributions

Die Möglichkeiten zur aktiven Beteiligung an Open-Source-Entwicklung ist so vielfältig wie die Anwendungsfälle von Software insgesamt. Grundsätzlich zählen zu einer Beteiligung alle Aktivitäten, die über den reinen Konsum der Software hinaus gehen.

- Das beginnt mit der Unterstützung anderer User zum Beispiel auf einer Mailingliste
- Feedback an die Entwicklerinnen und Entwicklern über Fehler oder Änderungswünsche
- Qualifizierte Fehlermeldungen möglicherweise sogar mit Patches zur Korrektur
- Testen von neuen Versionen
- Erstellung, Erweiterung und Übersetzung von Dokumentation
- Paketieren der Software für bestimmte Distributionen
- Implementieren von neuen Methoden, Modulen, Features
- Review von Code anderer Kontributorinnen und Kontributoren
- bis hin zur umfänglichen Verantwortung für die Teile des Projekts
- und dem Initiieren und Treiben neuer Open-Source-Projekte

> Alle Open-Source-Projekte sind mindestens partiell an der aktiven Beteiligung durch die Communities interessiert.

Alle Open-Source-Projekte sind mindestens partiell an der aktiven Beteiligung durch die Communities interessiert. Erfolgreiche Projekte zeigen sich deshalb einladend und offen, selbst wenn sie primär ein firmengesteuertes Entwicklungsmodell verfolgen.

Gleichzeitig haben erfolgreiche Open-Source-Projekte aber in der Regel eine Führungs- und Kontrollstruktur und es gibt Richtlinien für Form und Inhalt der Mitarbeit. So eine Struktur kann wie oben beschrieben aus einem firmengesteuerten Entwicklungsmodell bestimmt werden.

Sie kann aber zum Beispiel auch im Rahmen einer Foundation demokratisch verfasst sein, oder sie ist informell in Form einer Meritokratie gewachsen. Speziell bei den oben beschriebenen Projekten mit informeller Governance ist es vor der Herstellung und Kontribution größerer Beiträge ratsam, eine Abstimmung mit dem Projekt zu suchen.

4.2.1 Open-Source-Beteiligung aus Wirtschaftlicher Perspektive

Um den oben angesprochenen Mehraufwand bei der Pflege eines eigenen Zweigs (Branch) eines Open-Source-Projektes zu vermeiden, gibt es gute wirtschaftliche Gründe, auch erhebliche Beiträge in das Ursprungsprojekt einzubringen und so der Allgemeinheit zur Verfügung zu stellen.

Schon kleine Beiträge haben einen wirtschaftlichen Nutzen für die beitragenden Unternehmen. Feedback an die Entwicklerinnen und Entwicklern dient der Motivation und kann günstigstenfalls die Entwicklung in eine gewünschte Richtung lenken.

Indem Patches an die Upstream Community abgegeben werden, wird nicht nur der Aufwand zur Pflege eines eigenen Softwarezweigs gespart, in den dann die Upstream Patches zurückportiert werden müssen. Häufig werden durch so eine Contribution auch die darin realisierten Ideen von der Community aufgegriffen und selbständig weiterentwickelt. Im Idealfall reicht also auch hier ein kleiner Anstoß um große Veränderungen auf den Weg zu bringen.

Es gibt weitere wirtschaftlich berechenbare Werte und Nutzen bei der Beteiligung an Open-Source-Projekten.

- Im »Beauty Contest« der Unternehmen um die besten Talente ist die aktive Beteiligung an Open-Source-Projekten und einen Open-Source-Contribution-Policy im Unternehmen ein attraktives Unterscheidungsmerkmal.
- Die Unterstützung anderer User kann die eigene Kompetenz als Dienstleister in der Domäne einer Open-Source-Applikation unterstreichen.
- Mit Open-Source-Referenz-Implementationen lassen sich Standards setzen, es lässt sich Interoperabilität herstellen und es kann Vertrauen zwischen sonst konkurrierenden Parteien erzeugt werden.

In unserem komplexen globalen Wirtschaftsgeschehen können die sich ständig wandelnden Anforderungen an Informations- und Datenverarbeitung der handelnden Unternehmen weder durch Standardsoftware traditioneller Anbieterinnen und Anbietern noch durch Eigenentwicklung ausreichend bedient werden.

Digitale Transformation und damit Zukunftsfähigkeit kann nur durch Kollaboration und Kooperation mit Open-Source-Methodik und Open-Source-Software gelingen.

4.3 Collaboration-Tooling

Open-Source-Entwicklung ist charakterisiert durch die Kooperation ansonsten organisatorisch nicht verbundener Individuen über große räumliche und damit auch zeitliche Distanz. In diesem Setting kommt den technischen Kollaborations-Werkzeugen eine wichtige Rolle zu. Konkrete Beispiele oder gar Empfehlungen für Werkzeuge abzugeben, sprengt den Rahmen dieses Leitfadens. Jedes Projekt stellt sich seinen eigenen Werkzeugkasten zusammen. Dennoch möchte wir hier ein paar grundsätzliche Überlegungen und Erfahrungen teilen.

4.3.1 Kommunikation

Asynchrone und gleichzeitig flüssige und nachvollziehbare Kommunikation ist essentiell für die Kollaboration. Je größer und erfolgreicher ein Projekt sich entwickelt, desto mehr Teilnehmerinnen und Teilnehmern müssen häufig auch über mehrere Zeitzonen in die Kommunikation eingebunden werden.

Als universelle Selbstverständlichkeit kommen Email (speziell auch in Form von Mailinglisten) und Chat für die Kommunikation zum Einsatz. Typischerweise gibt es zu den relevanten Open-Source-Projekten auch eine Webseite (Homepage) auf der sich das Projekt vorstellt und weiterführende Information zum Beispiel in Form eines Wiki, eines Blog oder auch ein Diskussionsforum für die Nutzerinnen und Nutzern anbietet.

Für die Planung eines neuen Open-Source-Projektes ist das insofern wichtig, als die Aufwände zum Beispiel für die Moderation einer Mailingliste oder die Pflege einer Projektseite nicht unerheblich sind, bei der initialen Schätzung aber gerne vergessen werden.

Asynchrone und gleichzeitig flüssige und nachvollziehbare Kommunikation ist essentiell für die Kollaboration.

4.3.2 Werkzeugkette

Versionskontrolle: Da es bei der Open-Source-Entwicklung im Kern um Quelltexte von Software geht, nimmt das Versionskontrollsystem (VCS) eine zentrale Rolle ein. Hier geht es neben der Nach- und Rückvollziehbarkeit von Änderungen auch um die Koordination paralleler Entwicklungsstränge (Branch, Fork, Merge, etc). Neben historisch weiter existierenden VCS wie CVS, RCS, SCCS und Subversion (SVN) gibt es eine große Zahl proprietärer Angebote und es gibt Git.

Die Auswahl des VCS ist insofern relevant, als mit einem Wechsel typischerweise der Verlust von Metadaten und Historie einhergeht. Für aktuelle und insbesondere für neue Open-Source-Projekte ist wohl Git als Werkzeug und Datenformat der allgemeine Standard. Git ist ein verteiltes System und es gibt mehrere gut etablierte Plattformen, die zentrale Git-Server für Open-Source-Projekte kostenlos im Internet anbieten.

Automatisierung und Integration: Moderne Softwareentwicklung nutzt vielfältige Werkzeuge, Methoden und integrierte Umgebungen mit denen operative Elemente des kreativen Prozesses vereinfacht oder ganz automatisiert werden können.

Beispiele für solche Werkzeuge sind IDE, Modul-Bibliothek, Artefact Repository, Build Pipeline, Test Automation und viele mehr.

Die konkrete Ausprägung so einer Entwicklungsumgebung hängt unter anderem von der gewählten Programmiersprache und dem Deployment- und Betriebsformat der resultierenden Software ab. Für ein klassisch kompiliertes C-Programm sieht die Umgebung anders aus als für eine J2E Java Applikation oder für einen containerisierten Go Microservice. Für alle Aufgaben und Bereiche stehen leistungsfähige Open-Source-Werkzeuge zur Verfügung. Eine tiefere Betrachtung der vielfältigen Optionen für die Gestaltung solcher Entwicklungsumgebungen sprengt den Rahmen dieses Leitfaden.

> Werkzeuge können den sozialen Charakter der Interaktion unterstützen, ersetzen können sie ihn nicht.

Für jedes einzelne Open-Source-Projekt ist es in jedem Fall sinnvoll die prototypische Entwicklungsumgebung zusammen mit den Coding Standards zu beschreiben.

Plattformen: Bei der Auswahl der geeigneten Plattform geht es dann auch um die Rollen- und Rechteverwaltung (Commit, Review + Merge, Comment et cetera), das Management von Issues, die Automatisierung von Workflow und die Integration in verschiedenste Entwicklungsumgebungen.

Mit der Kontrolle über diese Funktionen geht eine strukturelle Macht über essenzielle Prozesse in dem Projekt einher. Werkzeuge können den sozialen Charakter der Interaktion unterstützen, ersetzen können sie ihn nicht.

Die Verwendung auch von kostenlosen Angeboten für zentrale Source Code Repositories und Versionskontrollsysteme hat einen Preis. Damit gehört auch die Frage, wer diesen Preis bezahlt und wie nachhaltig und sicher dieses Geschäftsmodell ist, in die Überlegungen zur Auswahl des geeigneten Werkzeugs.

4.3.3 Kreativität

Über die genannten Zwecke hinaus gibt es ein weites Feld an Technologie zur Unterstützung von kreativer Zusammenarbeit. Eine gute Übersicht von kreativen Methoden für Teams gibt die ↗Open Practice Library.

Und es gibt Werkzeuge wie Diagrammeditoren, Whiteboards, Kanban Boards, ↗Liquid Democracy, um diese Methoden anzuwenden und den kreativen Prozess zu unterstützen.

4.4 Fazit

Die hier aufgezeigten Modelle und Wege zur Kontribution kommen von sehr unterschiedlichen Situationen und werden von verschiedensten Absichten und Motiven getrieben.

Open-Source-Software kommt überall zum Einsatz: Im Smartphone, als Webservice oder als Infrastruktur in der Cloud. Auch in proprietären Softwareprodukten werden fast immer irgendwelche Komponenten, Bibliotheken, Tools oder Frameworks aus der Open-Source-Welt genutzt. Dem oben beschriebenen Bild der Kontributions-Pyramide entsprechend sollte sich alle Unternehmen mindestens ihrer Rolle als Nutzende von Open-Source-Software bewusst werden. Diese Rolle aktiv zu gestalten und zum Beispiel die aktive Beteiligung an Communities mit Erfahrungsberichten, Bugreports oder kleinen Patches ausdrücklich zu erlauben, ist ein erster Schritt.

> Es ist auch wichtig, als Nutzer die aktiven Rechte von Open-Source-Software zu kennen und wahrzunehmen.

Es ist auch wichtig, als Nutzer die aktiven Rechte von Open-Source-Software zu kennen und wahrzunehmen. Da, wo heute in proprietärer Software bereits ein Großteil der unterliegenden Technologien und Komponenten als Open Source zur Verfügung steht, ist es möglicherweise sinnvoll, aktiv in einen kollaborativen Entwicklungsprozess einzusteigen und das Portfolio an Open-Source-Lösungen zu vergrößern.

Bei Erstellung eines neuen Open-Source-Projekts gibt es, wie oben beschrieben, viele Freiheitsgrade, die, wenn sie richtig gewählt werden, einen starken Einfluss auf den Erfolg des Projekts haben können. Gerade eine gut gewählte Projekt-Governance, beispielsweise die Nutzung eines bewährten Open-Governance-Ansatzes oder einer von einer Foundation vorgegebenen Methodik, erleichtert es interessierten Firmen, sich an einem Projekt zu beteiligen. Dies resultiert aus dem Fakt, dass die Mechanismen etabliert und bewährt sind. Die Risiken sind somit wesentlich einfacher zu bewerten. Außerdem sind sie den prüfenden Rechtsanwälten bekannt, gegebenenfalls sogar schon geprüft. Das reduziert das Risiko einer Teilnahme am Projekt und macht dieses attraktiv.

5 Geschäftsmodelle rund um Open-Source-Software

Ein Grundprinzip von Open-Source-Software ist, dass für die Nutzung keine Lizenzgebühren anfallen dürfen. Diese Bedingung ist in der Open-Source-Definition der OSI verankert. Darüber hinaus aber – und das ist der entscheidende Punkt – ist **jede Vergütungsvereinbarung für jeden denkbaren Dienst für und mit Open-Source-Software zulässig**. Software, die per Lizenz nicht kommerziell genutzt werden darf, ist nach Definition der Open Source Initiative (OSI) keine Open-Source-Software.

Welches Geschäftsmodell man sich auch immer ausdenkt, ein Aspekt zeichnet den kommerziellen Umgang mit Open-Source-Software aus: Damit kann nur mittelbar ein Alleinstellungsmerkmal aufgebaut werden. Die Zugänglichkeit von Open-Source--Software und die mit ihr verknüpften Freiheitsrechte erleichtern das Entstehen von Alternativen. Jede Person, die geschäftlich mit Open-Source-Software arbeitet, sollte die Software, auf der ihr Service basiert, ihren Kundinnen und Kunden im vollen Bewusstsein bereitstellen, dass sie als Lieferant – bezogen auf die Software – leicht ersetzbar ist. Demzufolge wird der Mehrwert im Bewusstsein der eigenen besonderen Kompetenz erbracht und daraus das Alleinstellungsmerkmal geformt. Die Nutzung der Open-Source-Software kann dabei ein verkaufsfördernder Vorteil sein: Open-Source-Services geben Kundinnen und Kunden einen größeren Grad an Freiheit im Verhältnis zu ihrem Lieferanten. So gesehen war und ist Open-Source-Software immer schon ein besonderes marktwirtschaftliches Instrument.

Open-Source-Software ist also aus sich heraus kein Geschäftsmodell, sondern »nur« eine besondere, kooperative Entwicklungsmethode. So begleitet die Entwicklung der Szene durchgängig die Frage, wie genau solche Geschäftsmodelle mit einer Ausrichtung auf Open Source aussehen können. Von Unternehmensseite wurde beispielsweise lange bezweifelt, ob für die »freie« Software professioneller kommerzieller Support überhaupt möglich sei. Fraglich war zunächst auch, ob es wirtschaftlich überhaupt sinnvoll sein kann, Open-Source-Software über die reine Nutzung hinaus auch in Form von Entwicklungsaufträgen oder über eigene Beteiligung zu unterstützen. Umgekehrt zweifelten Open-Source-Communities, ob sich Firmen an die Spielregeln der freien Software halten würden und ob eine kommerzielle Verwertung sich mit dem Geist der Gemeinschaften würde vereinbaren lassen.

> Die Zeit, in der Open Source ein Spezialthema war und zum Beispiel auf eigenen Fachmessen präsentiert wurde, sind vorbei.

Davon kann heute keine Rede mehr sein. Open-Source-Software ist integraler Bestandteil vieler kommerzieller Produkte und insbesondere im Bereich der Cloud-Services ist Open Source der einzige Weg, um die Skalierbarkeit, die solche Dienste notwendigerweise auszeichnet, in wirtschaftlich sinnvoller Art umzusetzen. Spektakuläre Übernahmen und Bewertungen von Firmen, die ihr Geschäftsmodell primär auf Entwicklung von Open-Source-Software ausrichten, bestätigen die wirtschaftliche Bedeutung des Konzepts Open Source.

Die Zeit, in der Open Source ein Spezialthema war und zum Beispiel auf eigenen Fachmessen präsentiert wurde, sind vorbei. Heute ist Open Source implizit oder explizit Teil aller Bereiche der Wirtschaft, in denen Software relevant ist. Das zeigt sich auch in

der Organisation in Verbänden: So existiert die Open Source Business Alliance (OSBA)[31] und auch innerhalb des Bitkom e.V. gibt es den etablierten Arbeitskreis Open Source[32].

Die **existierenden Geschäftsmodelle** lassen sich in der Hinsicht **klassifizieren**, ob das Geschäftsmodell die Open-Source-Software als Mittel zur Erbringung eines davon unabhängigen Geschäftszwecks nutzt (Geschäftsmodelle **mit** Open-Source-Software) oder ob das Geschäftsmodell unmittelbar um die Open-Source-Software aufgebaut ist (Geschäftsmodelle **für** Open-Source-Software). Dieser Klassifizierung folgend werden in den Unterkapiteln Beispiele für Geschäftsmodelle beschrieben, die so auf dem Markt beobachtbar sind. Diese Auflistung erhebt nicht den Anspruch auf Vollständigkeit, gibt aber ein umfassendes Bild über die hauptsächlich beobachtbaren Modelle.

31 vgl. http://www.osb-alliance.de/
32 vgl. https://www.bitkom.org/Bitkom/Organisation/Gremien/Open-Source.html

5.1 Geschäftsmodelle mit Open-Source-Software

Bei diesen Geschäftsmodellen wird Open-Source-Software genutzt, um damit ein Produkt zu entwickeln oder einen Service zu erbringen. Die Open-Source-Software erfüllt hier Mittel zum Zweck und wird häufig nur genutzt. Viele Firmen nutzen den Mechanismus aber auch, um gezielt nicht wettbewerbsrelevante Bereiche ihres Software-Stacks durch Zusammenarbeit mit anderen Firmen effizienter zu entwickeln oder durch eine breitere Nutzerbasis qualitativ zu verbessern.

5.1.1 Services mit Open-Source-Software

In diesem Modell werden Services angeboten, die mittels Nutzung von Open-Source-Software erbracht werden, man spricht auch von Software-as-a-Service (SaaS). Soziale Netzwerke, Webshops, Suchmaschinen, Cloud-Provider, sie alle werden unter teilweise massivem Einsatz von Open-Source-Software betrieben. Kundinnen und Kunden bezahlen für die Erbringung des Service, oder die Services werden durch Geschäftsmodelle wie den Verkauf von Werbung finanziert. Ob das mit Open-Source- oder proprietärer Software umgesetzt wird, ist für die eigentliche Leistung unerheblich. Open-Source-Software fließt als begünstigender Kostenfaktor in das Geschäftsmodell ein.

Nicht zuletzt hat die Verfügbarkeit von Open-Source-Software massiv dazu beigetragen, Startups die Möglichkeit zu bieten, mit minimalen Investitionen auf viele höchst leistungsfähige Software-Stacks zuzugreifen und innovative Services mit einer bisher ungekannten Geschwindigkeit zu realisieren.

5.1.2 Open-Source-Software as a Service

Eine Sonderform der Services mit Open-Source-Software ist die Bereitstellung einer Open-Source-Software als Service im Internet. Hier kann zwischen Bereitstellung durch den Hersteller oder durch Dritte unterschieden werden.

Bei der Bereitstellung durch den Hersteller handelt es sich zum Beispiel um eine Open-Source-Software, die von einem Unternehmen zur Verfügung gestellt oder hauptsächlich entwickelt wird. Neben der frei verfügbaren Software bietet dieses Unternehmen aber auch einen Internet-Service an, der den Nutzer von der Installation und Pflege der Software entbindet und diese für ihn einfach verfügbar macht. Beispiele gibt es

im Bereich von Datenbanken oder virtuellen Kommunikationsplattformen und etwa bei Projekten der Cloud Native Computing Foundation[33].

Gerade für Open-Source-Software, die durch die freie Verfügbarkeit eine hohe Verbreitung erreicht hat, ist das Angebot als Service oft eine attraktive Form, Einnahmen zu erzielen, die die Pflege und Weiterentwicklung der Software finanzieren. Dieses Modell fällt dann allerdings eher in die Kategorie »Geschäftsmodelle für Open-Source-Software«.

Beim Betrieb durch Dritte wird die Software von einem Unternehmen zur Verfügung gestellt, das nicht als primärer oder alleiniger Entwickler fungiert. Beispiel hierfür ist die Bereitstellung von Services durch Cloud-Anbieter. Um dieses Modell hat es in den letzten Jahren einige Kontroversen gegeben, da manche Hersteller sich im Nachteil gegenüber den üblicherweise weit größeren Cloud-Anbietern gesehen und versucht haben, durch Änderung der Lizenzierungen diesen Nachteil auszugleichen. Die Lizenzen, zu denen diese Hersteller gewechselt sind, schließen typischerweise die Nutzung durch Konkurrierende beim Betrieb der Software aus und sind damit keine Open-Source-Lizenzen mehr. Open-Source-Lizenzen erfordern die diskriminierungsfreie Nutzbarkeit, auch durch die Konkurrenz. Man muß auch feststellen, dass die großen Cloud-Anbieter zu den größten Open-Source-Beitragenden gehören.[34]

Open-Source-Lizenzen erfordern die diskriminierungsfreie Nutzbarkeit, auch durch die Konkurrenz.

5.1.3 Produkte mit Open-Source-Software

Vergleichbar zu den Services kann Open-Source-Software natürlich auch in die Entwicklung von Produkten einfließen. Hier werden Open-Source-Komponenten in das Produkt eingebaut und stellen häufig einen großen Anteil an der Software des Produkts dar. Dies ist der klassische Use-Fall, Open-Source-Software wird in einem Softwarekontext genutzt und mit dem Produkt redistributiert. Ein augenscheinliches Indiz für die Relevanz dieses Modells stellen die Listen von Open-Source-Lizenzen dar, die in vielen Konfigurationsmenüs oder Handbüchern von Produkten wie Fernsehern, Routern, Computerspielen und ähnlichem zu finden sind.

5.1.4 Open-Source-Software als Enabler für andere Geschäftsmodelle

Bei diesem Geschäftsmodell dient Open-Source-Software dazu, die Nutzung eines Services oder die Erbringung eines anderen Geschäftszwecks zu vereinfachen. Unternehmen mit diesem Modell erstellen durchaus recht umfangreich Open-Source-Software. Das können zum Beispiel ein Betriebssystem für Smartphones, ein Browser oder aber Entwicklungstools sein. Durch die verbreitete Nutzung dieser Software können

33 siehe viele Projekte der CNCF Cloud Native Interactive Landscape, vgl. https://landscape.cncf.io/
34 vgl. https://opensourceindex.io/

dann so unterschiedliche Geschäftsmodelle, wie das Sammeln von Daten für Werbung oder die verstärkte Nutzung von bereitgestellten, kostenpflichtigen Cloud-Diensten, unterstützt werden. Das Investment in die Open-Source-Software dient also nur mittelbar dem Geschäftszweck, bietet aber Kontrollpunkte mittels derer das eigene Geschäftsmodell vorangetrieben werden kann.

5.2 Risikobetrachtung bzgl. der Nutzung von Open-Source-Software

Betrachtet man die beschriebenen Geschäftsmodelle, so steht die Nutzung von zur Verfügung gestellter Open-Source-Software im Vordergrund. Ein Anbieter profitiert von der Entwicklung der Open-Source-Community, verlässt sich aber weitgehend auch darauf, dass die Community die Software im gewollten Sinne weiterentwickelt. Dies stellt ein Risiko dar, welches vom nutzenden Unternehmen bewertet werden muss. Gerade bei einer großen Abhängigkeit von einzelnen Open-Source-Komponenten muss ein nutzendes Unternehmen Maßnahmen zur Risikokontrolle installieren.

Gerade bei einer großen Abhängigkeit von einzelnen Open-Source-Komponenten muss ein nutzendes Unternehmen Maßnahmen zur Risikokontrolle installieren.

Auf der anderen Seite erzwingt die heutige Marktdynamik, die gerade auch durch den Open-Source-Mechanismus entstanden ist, das Überdenken der Strategie wie nicht-wettbewerbsrelevante Anteile des eigenen Produkt- oder Serviceportfolios entwickelt und über den gesamten Lebenszyklus zur Verfügung gestellt werden. Das Risiko besteht dann darin, dass zu langes Festhalten an einer proprietären Eigenentwicklung zu viele Entwickler bindet und die Innovationsgeschwindigkeit des Unternehmens leidet. Eine Risikokontrolle beinhaltet ein erhöhtes Investment in Open-Source-Software, zum Beispiel durch eine Beteiligung an oder auch Gründung von Open-Source-Communities.

Will sich ein Unternehmen verstärkt an Open-Source-Projekten beteiligen, stehen zwei Optionen zur Verfügung. Zum einen die Beauftragung Dritter, die dann aktiv an Open-Source-Communities teilnehmen. Die dazu passenden Geschäftsmodelle werden im nächsten Unterkapitel beschrieben. In diesem Abschnitt werden die Optionen für eine direkte Beteiligung an Open-Source-Projekten beschrieben.

An dieser Stelle ist es sinnvoll, zwischen eher allgemeiner Plattform-Software und vertikaler Software zu unterscheiden. Plattform-Software ist typischerweise nicht domänenspezifisch, sondern allgemeiner Natur. Als Beispiel dienen Betriebssysteme, die zum Betrieb eines Devices benötigt werden, die aber keine zur Kernkompetenz des Unternehmens zählende Funktionalität zur Verfügung stellen. Im Gegensatz dazu geht es bei vertikaler Open-Source-Software als Produkt- oder Servicebestandteil darum, in einem konkreten Markt typische, aber nicht wettbewerbsrelevante Software in einer gemeinsamen Zusammenarbeit von Marktteilnehmenden zu entwickeln. Beispiele für diese Form sind die Networking Working Group der Linux Foundation[35] im Bereich Telekommunikationsnetzwerke oder die Academy Software Foundation[36] im Bereich Motion Picture.

35 vgl. https://www.lfnetworking.org
36 vgl. https://www.aswf.io

5.2.1 Beteiligung an Plattform Open-Source-Software

Für allgemeine Software-Projekte ist eine typische Strategie, sich zunächst sporadisch zu beteiligen. Das kann in Form von Bugfixing, kleinerer Featurebeiträge oder auch nur Community-Aktivitäten wie Bugreporting und Beteiligung an Diskussionen passieren. Je nach Risikobewertung geht es hier meist darum, dringende Mängel der Software aus der Perspektive der eigenen Nutzung zu beheben. Am Beispiel Betriebssysteme und hier im Speziellen am Beispiel Linux sieht man aber auch sehr gut, dass eine Beteiligung recht schnell tiefer gehen kann. Für Unternehmen, die eigene Hardware wie Sensoren oder Aktuatoren bereitstellen, besteht die Notwendigkeit, ihre Hardware für Linux mittels Treiber nutzbar zu machen. In Linux sind Treiber Teil des Linux-Kernels, damit dieser bei der Weiterentwicklung des Kernels nicht inkompatibel wird und für die Nutzenden der Hardware einfach verfügbar ist. In der Konsequenz ist es also naheliegend für die Hardwarehersteller, Software im Linux-Kernel beizutragen.

Für die meisten Unternehmen ist diese Form der Kollaboration mit Open-Source-Communities davon getrieben, sich in etablierten Ökosystemen einzubringen und die bereits existierende Standardisierung durch die Open-Source-Software zu unterstützen. Seltener geht es darum, auf dieser allgemeinen Ebene eine neue Standardisierung durch eine Open-Source-Software voranzutreiben. Aber auch das ist nicht abwegig, als Beispiel sei hier Zephyr[37] erwähnt, ein Betriebssystem für ressourcenarme Geräte, welches erst in den letzten Jahren aus Industriekollaboration entstanden ist und von einer Vielzahl von Unternehmen aus unterschiedlichen Bereichen unterstützt wird.

5.2.2 Beteiligung an vertikalen Open-Source-Projekten

Für viele Unternehmen wird diese Beteiligung ein interessantes strategisches Modell sein, um Entwicklungsressourcen aus nicht wettbewerbsrelevanten Software-Anteilen zu freizusetzen. Dies ermöglicht die Konzentration auf diversifizierende Software-Anteile und damit eine Stärkung der Marktposition.

Die Idee der vertikalen Open-Source-Projekte besteht darin, dass Software, die aktuell proprietär entwickelt wird, weil sie zur Erbringung des Geschäftsmodells notwendig ist, in dieser oder leicht abgewandelter Form aber von anderen Marktteilnehmern ebenfalls bereitgestellt werden muss. In diesem Fall besteht das Potential, diese Software-Anteile in einer Zusammenarbeit mit anderen Marktteilnehmenden gemeinsam zu entwickeln und auf diesem Weg für die Industrie zu standardisieren. Durch die gemeinsame Entwicklung kann mittelfristig die eigene Investition in die Software kleiner werden und somit der erwünschte Effekt der Freisetzung von Entwicklern erzielt werden.

37 vgl. https://www.zephyrproject.org

Die oben erwähnten Beispiele zeigen sehr gut die Motivation zur Zusammenarbeit. Im Telekommunikationsbereich geht es um eine höchst zuverlässige Bereitstellung des Netzwerks. Es kommt eher auf die Qualität der genutzten Hard- und Software an, als auf umfangreiche Diversifizierungsmöglichkeiten, ideal für eine Zusammenarbeit im Open-Source-Bereich. Auch im Motion-Picture-Umfeld entsteht die Diversifikation durch die Kreativität der Beteiligten. Die Bereitstellung der Tools stellt eher ein Hindernis dar, weil diese eine hohe, für die beteiligten Unternehmen häufig kaum leistbare, Investition bedeuten, auf der anderen Seite aber die Effizienz der Nutzbarkeit im Vordergrund steht, sprich die Effizienz der Kreativen.

5.3 Services für Open-Source-Software

Der Nutzung von Open-Source-Software stehen die Geschäftsmodelle gegenüber, bei denen Services zum adäquaten Betrieb von Open-Source-Software vergütet werden. Die anbietenden Firmen müssen dabei nicht zwingend selbst Open-Source-Software entwickeln. Für Anbietende eigener Open-Source-Software (Applikationsanbietende) können weitere Serviceangebote ein sekundäres Gewinnmodell sein. Man kann beobachten, dass generell bei diesen Geschäftsmodellen verschiedenste Abrechnungsmetriken zum Einsatz kommen. Beispiele sind Kontingent-Verrechnung oder Nutzungszahlen-basierte Subscriptions. Die Services lassen sich wie folgt typisieren:

5.3.1 Support

Hierunter fällt sowohl die technische als auch die nicht-technische Kundenbetreuung, die in der Regel über einen Helpdesk oder ein Call-Center verwirklicht wird. Supportdienstleistende können sich auf die technischen Aspekte der Installation und Integration neuer Software in bestehende Systeme spezialisieren, ebenso aber Unterstützung bei alltäglichen Anwendungsproblemen bieten.

Häufig werden Supportverträge abgeschlossen, die verschiedene Supportstufen, Verfügbarkeiten der Beratung und Reaktionszeiten festlegen. Supportstufen umfassen dabei First-, Second- und Third-Level-Support, jede Stufe stellt eine weitere Eskalation innerhalb des Systems dar. Variation gibt es typischerweise auch bei der Abrechnung. Die Bandbreite geht dabei von Einzelabrechnung der Supportanfragen bis hin zu Subskriptionen, die für die vereinbarte Vertragsdauer Supportanfragen ermöglichen.

5.3.2 Entwicklung

Bei der kundenspezifischen Weiter- bzw. Auftragsentwicklung integriert der Dienstleistende bestimmte Funktionen oder Gestaltungsmerkmale in die Open-Source-Software. So entstehen beispielsweise auf Basis eines bereits existierenden Softwarestandes eigenständige Anwendungen, die auf die Auftraggebenden zugeschnitten sind. Ein solches Angebot für Open-Source-Software ermöglicht auch, unabhängig von einer zum Beispiel Community-getriebenen Roadmap spezielle Features (früher) zu verwirklichen, die für eine Leistung mit Open-Source-Software benötigt werden. Häufig, aber nicht zwangsläufig, werden diese kundenspezifischen Entwicklungen unter einer

Open-Source-Lizenz veröffentlicht und können bei hinreichend vorhandenem Interesse Zugang in das ursprüngliche Open-Source-Projekt finden.

Bezogen auf das Modell »Open-Source-Software« besteht der Service unter Umständen nicht nur in der reinen Weiterentwicklung der Software, sondern auch darin, den Kontakt mit der Open-Source-Community, die Kooperation oder die Rückveröffentlichung von Entwicklungsständen zu organisieren.

5.3.3 Betrieb beziehungsweise Bereitstellung

Im Gegensatz zum Open-Source-Software-as-a-Service-Modell geht es bei diesem Geschäftsmodell darum, für einen spezifische Kunden die Open-Source-Software so aufzubereiten, dass dieser die Software direkt nutzen kann. Dies beinhaltet Services wie den Betrieb eines Services auf kundenspezifischer Hardware, Integration der Software und Bereitstellung mittels des kundenspezifischen Software-Management-Mechanismus, um die Software auf den Rechnern des Kunden zur Verfügung zu stellen. Auch die Konfiguration der Software für den Kunden wird in diesem Modell von den Zuliefernden übernommen. Dieses Modell wird typischerweise mit anderen Ausprägungen aus diesem Unterkapitel kombiniert.

5.3.4 Wartung

Wartung (Maintenance) von genutzter Software ist meist festgelegt auf einen bestimmten Zeitraum (beispielsweise ein Jahr) sowie auf einen definierten Leistungsumfang (wie das Einspielen regelmäßiger Sicherheitsupdates). Auch nicht mehr aktiv weiterentwickelte Software wird oftmals noch in komplexeren IT-Architekturen eingesetzt und muss gewartet werden. Auch nach der offiziellen Einstellung des Open-Source-Projektes können diese Aufgaben von Drittanbietenden übernommen werden, um eine Software oder Infrastruktur lauffähig zu halten und den Nutzungszeitraum zu verlängern.

5.3.5 Beratung

Beratungsangebote können alle Punkte des Lebenszyklus umfassen, beispielsweise Studien, Analysen und Konzeptionen. Sie zielen sowohl auf die Phase vor der Einführung einer Software (wie Auswahl und Evaluierung oder Ausschreibung) als auch auf deren Begleitung, um die Übergangszeit sowie den späteren Einsatz erfolgreich zu gestalten und Unternehmensprozesse anzupassen (zum Beispiel Erstellung und Umsetzung eines Sicherheitskonzeptes). Ebenso finden sich Beraterinnen und Berater, die Firmen bei deren eigenen Software-Projekten betreuen. Das kann sinnvoll sein, wenn erstmals Open-Source-Software veröffentlicht werden soll sowie beim Umgang mit Open-Source-Communities oder Lizenzbedingungen.

5.3.6 Zertifizierung

Ein weiteres Geschäftsmodell basiert auf der Tatsache, dass Open-Source-Software in der Regel ohne Garantien kommt. Da Open-Source-Software in immer weitere Bereiche vordringt, werden Sekundäreigenschaften wie Safety oder Security bedeutender. In Domänen wie der Automobilbranche oder in der Luftfahrt gibt es ISO-Standards, deren Einhaltung durch die Software nachgewiesen werden muss, um eine Genehmigung für den Betrieb des Produkts inklusive seiner Software zu erhalten.

Da Open-Source-Software in immer weitere Bereiche vordringt, werden Sekundäreigenschaften wie Safety oder Security bedeutender.

In diesem Geschäftsmodell werden Services angeboten, die eine Qualifizierung der Open-Source-Software entsprechend einer der ISO-Normen für einen konkreten Use Case durchführen. Mit dieser Zertifizierung wird die Nutzbarkeit von Open-Source-Software in entsprechenden Bereichen ermöglicht. Als Beispiel an dieser Stelle sei das ELISA Project der Linux Foundation[38] erwähnt, in dem es um die Nutzung von Linux in safety-kritischen Anwendungen geht. In diesem Projekt geht es um Prozesse und Methoden, um eine solche Nutzung zu ermöglichen. Dies bildet im Nachgang die Möglichkeit auf Basis der standardisierten Prozesse Zertifizierungsdienstleistungen anzubieten.

5.3.7 Schulung

Schulungen vervollständigen das Angebot um Open-Source-Software herum. Zahlreiche Anbietende – vor allem kleine und mittlere Unternehmen (KMU) oder Einzelpersonen – haben sich darauf spezialisiert, eine breite Palette von Kursen und Zertifizierungen anzubieten. Zielgruppe sind sowohl Anwender als auch Administratoren und Programmierende. Unternehmen können ihre Mitarbeitenden gezielt und professionell betreut weiterbilden und damit internes Know-how mehren. Auch große Open-Source-Software-Anbietende zählen Schulungen zu ihrem Repertoire: Dadurch verbreiten und verbreitern sie das Wissen um ihr Open-Source-Angebot und bieten zeitgleich Service für Kunden.

5.3.8 Duale Lizenzierung

Prinzipiell gibt es viele Spielarten der Dualen Lizenzierung[39]. In diesem Kapitel soll dabei ausschließlich die Kombination aus Open-Source- und proprietärer Lizenz betrachtet werden. Hierbei wird eine Software mittels einer typischerweise eher restriktiven Open-Source-Lizenz zur Verfügung gestellt. Kann oder will der Nutzer der Software diese unter den Bedingungen der Open-Source-Lizenz nicht einsetzen, kann er nötige Nutzungsrechte an einer proprietären Version gegen Entgelt erwerben.

38 vgl. https://elisa.tech

39 Grundsätzlich gibt es diese Art der Lizenzierung auch rein Open-Source-intern: Dann geht es den Entwicklern einer Software darum, über die mehrfache Lizenzierung unter verschiedenen Open-Source-Lizenzen eine besondere Sicherheit im Hinblick auf die Lizenzkompatibilität zu gewährleisten.

Dies kann für beide Parteien sinnvoll sein: Der Software-Kunde muss die normalerweise mit der Software verbundenen Open-Source-bedingten Verpflichtungen, zum Beispiel einen Copyleft-Effekt, nicht erfüllen. Der Softwarehersteller wiederum kann spezielle Features, deren Entwicklung er nicht über den Verkauf von Services abzudecken vermag, direkt den Nutzenden in Rechnung stellen.

Ein Spezialfall der Dualen Lizenzierung ist das Open-Core-Modell. Hierbei wird nicht die komplette Software unter einer je nach Version Open-Source-oder proprietärer Lizenz zur Verfügung gestellt, sondern ein Teil der Software wird unter einer OSI-zertifizierten Open-Source-Lizenz distribuiert und ein anderer Teil unter einer proprietären Lizenz. Oft ist hierbei eine allgemein nutzbare Grundversion als Open-Source-Software verfügbar und zusätzliche Komponenten, zum Beispiel zur Integration in ein Enterprise-Umfeld, können gegen Zahlung von Lizenzgebühren als proprietäre Software erworben werden. Eine Herausforderung dieses Modells ist, dass die Entscheidung, welche Komponente in welche Version geht, oft nicht leicht fällt und prinzipbedingt die jeweils andere Nutzergruppe benachteiligt.

5.4 Weitere Modelle

Neben den beschriebenen »klassischen« Geschäftsmodellen existiert eine Vielzahl von weiteren Modellen, von denen wir hier nur eine kleine Anzahl kurz beleuchten.

5.4.1 Spendenbasiertes Finanzierungsmodell

Viele Open-Source-Entwickler arbeiten ehrenamtlich und erstellen ihre Software nicht für eine Firma oder in anderem geschäftlichen Kontext, sondern als freiwillige Leistung. Die Motivation dafür kann zum Beispiel Neugierde, Lernen, Reputation, Spaß am Experimentieren oder die Einbindung in eine Community sein.

Manche Nutzenden möchten auch solchen Freiwilligen eine monetäre Zuwendung zukommen lassen. Gerade bei Projekten, die eine größere Verbreitung gefunden haben und auch kommerziell eingesetzt werden, kann es auch im Interesse von Firmen, die die Software nutzen, sein, nachhaltigere Entwicklung durch finanzielle Unterstützung zu ermöglichen.

Viele Open-Source-Entwickler arbeiten ehrenamtlich und erstellen ihre Software nicht für eine Firma oder in anderem geschäftlichen Kontext, sondern als freiwillige Leistung.

Hierfür bietet sich ein spendenbasiertes Finanzierungsmodell an. Das kann zum Beispiel über eine Foundation erfolgen, die als finanzieller Stellvertreter für eine Freiwilligen-Community agiert, oder über Plattformen, die es ermöglichen, Spenden an spezifische Projekte oder Entwicklerinnen und Entwickler zu geben. Es gibt davon inzwischen eine Vielzahl von Varianten, sei es als Crowd-Funding, als Subscription oder integriert in eine Code-Hosting-Plattform.

5.4.2 Foundation-Modell

Die großen Foundations bieten ein spezielles Zusammenarbeitsmodell an. Durch ihre Rechtsform und Policies ermöglichen sie Unternehmen eine offene und unabhängige Zusammenarbeit, die vor kartellrechtlichen Problemen schützt. In der Regel entstehen auf diesem Weg Open-Source-Projekte oder -Arbeitsgruppen, die durch einen eigenen Community-Vertrag geregelt werden. Diese Verträge beinhalten typischerweise einen Mitgliedsbeitrag, der von den teilnehmenden Firmen geleistet werden muss und eine Steuerungsorganisation, die dazu dient, das über die Mitgliedsbeiträge eingenommene Budget auf die Projekte aufzuteilen.

Prinzipiell kann niemand an einer Beteiligung an einem Open-Source-Projekt gehindert werden, durch die vertraglich geregelte Struktur wird aber eine sinnvolle Budgetierung

der Aktivität und eine zielorientierte Vorgehensweise sichergestellt. Dieses Modell kommt typischerweise bei den oben beschriebenen vertikalen Open-Source-Aktivitäten der beteiligten Unternehmen zum Einsatz, um die Zusammenarbeit rechtlich zu verankern.

Ein weiteres Geschäftsmodell, welches von den Foundations praktiziert wird, ist das Ausrichten von Konferenzen und User-Treffen. Hier geht es darum, den Open-Source-Communities Plattformen zum Austausch zu bieten und Nutzende mit Kontributoren zusammenzubringen.

6 Strategische Betrachtung von Open Source

6.1 Open-Source-Software im Unternehmen

Noch vor 15 Jahren galt der Einsatz von Open-Source-Software in Unternehmen als revolutionär. Firmen, die öffentlich erklärten, dass sie Open-Source-Software einsetzen, wurden argwöhnisch und misstrauisch beobachtet. Heute hingegen zeigen alle Studien[40] einheitlich, dass der Einsatz von Open-Source-Software in Unternehmen ganz normal geworden, ja geradezu nicht wegzudenken ist.

Es gibt vermutlich kein Unternehmen mehr, das keine Open-Source-Software einsetzt. Umso erstaunlicher ist es, dass Open-Source-Software in vielen Unternehmen immer noch eher auf technische und rechtliche Belange reduziert wird und nicht als neue Philosophie und kollaboratives Produktions-, Vertriebs- und Geschäftsmodell betrachtet wird. Dass dies der Bedeutung von Open-Source-Software bei Weitem nicht gerecht wird, zeigt die Tatsache, dass Open-Source-Software seit einigen Jahren die ITK-Ökosysteme revolutioniert. Dies geschieht oftmals leise und am Anfang unbemerkt – die Auswirkungen dieser Revolution sind trotzdem weitreichend und unübersehbar.

Die strategische und wirtschaftliche Wichtigkeit des Themas Open-Source-Software zeigen Entwicklungen der letzten Jahre, wie die Dominanz des Mobilen Betriebssystems Android, die Marktdurchdringung von Open-Source-Software-Frameworks für Künstliche Intelligenz, Investitionen und Akquisitionen von Firmen wie Redhat oder Github durch IBM und Microsoft. Unternehmen sollten alle Einflussfaktoren von Open-Source-Software berücksichtigen und regelmäßig überprüfen, ob ihre Einstellung und Strategie geschäftsfördernd ist oder nicht. Die Manifestierung dessen in einer Open-Source-Strategie ist empfehlenswert.

Die strategische und wirtschaftliche Wichtigkeit des Themas Open-Source-Software zeigen Entwicklungen der letzten Jahre.

40 Siehe zum Beispiel https://www.bitkom.org/opensourcemonitor

6.2 Open-Source-Strategie-entwicklung im Unternehmen

6.2.1 Grundüberlegung zu einer Open-Source-Strategie

Die folgenden Überlegungen sollen Unternehmen die Definition einer Open-Source-Strategie erleichtern und eine Hilfestellung bei der Ergreifung der notwendigen flankierenden Maßnahmen bieten. Ob alle genannten Betrachtungen, Analysen und Stakeholder in den Unternehmen durchgeführt werden oder vorhanden sind, hängt dabei sehr von der Größe des Unternehmens ab. Ausgangspunkt der Betrachtung mag eine gängige Definition des Begriffs »Strategie« sein:

»Unter Strategie werden in der Wirtschaft klassisch die (meist langfristig) geplanten Verhaltensweisen der Unternehmen zur Erreichung ihrer Ziele verstanden. In diesem Sinne zeigt die Unternehmensstrategie in der Unternehmensführung, auf welche Art ein mittelfristiges (ca. 2–4 Jahre) oder langfristiges (ca. 4–8 Jahre) Unternehmensziel erreicht werden soll. (...)

Im Zusammenhang mit der Unternehmensstrategie wird oft von den vorgeordneten Konzepten der Vision und des Unternehmensleitbildes gesprochen, sowie von Strategischem Management. Als nachgeordnet werden Teilstrategien (Marketingstrategie, Finanzierungsstrategie etc.) und die taktische (mittelfristige) sowie die operationale (kurzfristige) Ebene angesehen.«[41]

Bezogen auf Open-Source-Software ist zu bestimmen, inwieweit deren Einsatz, Mitarbeit an oder Veröffentlichung von eigenen Open-Source-Projekten dazu beitragen kann, die Geschäftsziele zu erreichen. Zur Ableitung einer solchen Strategie gilt es verschiedene Aspekte zu beachten.

Wesentlicher Ausgangspunkt bei der Definition einer Strategie zum Themenfeld Open-Source-Software sind die Vision, Ziele und generelle Strategie des Unternehmens. Abhängig hiervon sollte der Einsatz von Open-Source-Software als Stilmittel zur Erreichung der Unternehmensziele geplant werden und eine kongruente Open-Source-Strategie etabliert werden. Eine unterschiedliche Ausprägung je nach Produkt, Segment, Bereich, Einheit oder Gesellschaft eines Unternehmens mag je nach Größe und Diversität opportun oder gar zweckdienlich sein.

41 Vgl. die allgemeine Begriffserklärung in Wikipedia (https://de.wikipedia.org/wiki/Strategie_(Wirtschaft))

Open-Source-Software ist dabei nie als Selbstzweck, sondern als unterstützendes Mittel zur Erreichung der Unternehmensziele zu verstehen. Demzufolge ist die Open-Source-Strategie eines Unternehmens immer der Unternehmensstrategie untergeordnet.

6.2.2 Strategische Richtungen

Ein effizienter und effektiver Einsatz von Open-Source-Software kann in vier wesentliche Handlungsmaxime und somit Open-Source-Strategien unterteilt werden:

1. Weitestgehende Unterbindung von Open-Source-Software (Limit Open-Source-Software)
2. Förderung der Verwendung von Open-Source-Software (Use)
3. Förderung des Beitrags zu Open-Source-Software (Contribute)
4. Förderung der Erstellung von Open-Source-Software (Create)

Diese vier Strategierichtungen sollen nicht als in sich geschlossene und abgegrenzte, eigenständige Konzepte verstanden werden. Vielmehr zeigt sich, dass sich je nach Unternehmen und nach Reifegrad des Themas Open-Source-Software sowie der individuellen Bestrebungen und der erwarteten Vorteile die Übergänge zwischen diesen Strategien fließend sind. Ausgehend von einer ungemanageten Verwendung oder einer Unterbindung von Open-Source-Software über die bewusste Verwendung (Use) und Teilhabe (Contribute) hin zu der eigenständigen Erstellung von Open-Source-Software (Create).

6.2.3 Ziele einer Open-Source-Strategie

Treiber und Einflussfaktoren für die Definition einer Open-Source-Strategie können unter anderem sein:

- Setzen von Standards
- Marktpräsenz und Außenwirkung
- Gewinnung von Marktanteilen
- Bindung von Kunden und Lieferanten
- Talentgewinnung
- Wissensaneignung
- Kostenreduktion
- Sicherheitsanforderungen
- Markt- und Kundenanforderungen

6.2.4 Resümee und Notwendigkeit einer Open-Source-Strategie

Die Auswahl und konsequente Vorgabe einer Open-Source-Strategie und die Ausprägung und Reichweite innerhalb eines Unternehmens mag von den verschiedensten Faktoren abhängig sein. Es gilt jedoch sicher, dass kein Unternehmen mehr ohne Open-Source-Software auskommt. Insofern sollten zur Sicherstellung der gewünschten Ausschöpfung der Potenziale von Open-Source-Software sowie der etwaig notwendigen Reduktion von Risiken, die sich aus Open-Source-Software ergeben können, eine Strategie und damit verbunden die entsprechenden Handlungsmaßnahmen festgelegt werden. Umso bemerkenswerter ist vor diesem Hintergrund die Tatsache, dass mehr als 70 % der Unternehmen noch keine Open-Source-Strategie haben.[42]

Es ist zumindest eine minimale Open-Source-Strategie notwendig, die sicherstellt, dass Lizenzbedingungen eingehalten werden. Näheres dazu im ↗ Kapitel 7 »Compliance«.

Bei Beteiligung an Open-Source-Projekten und der Gründung eigener Open-Source-Projekte sind weitergehende strategische Überlegungen notwendig, um Fragen der Zusammenarbeit mit anderen Unternehmen, Governance für Projekte oder Community-Management zu beantworten. Näheres dazu im ↗ Kapitel 4 »Erstellen von Open-Source-Software«.

42 Siehe zum Beispiel https://www.bitkom.org/openSourcemonitor2019

6.3 Open-Source-Program Office (OSPO)

Als Teil einer Open-Source-Strategie gilt es festzulegen, wie das Thema Open Source und die damit verbundenen Aufgaben im Unternehmen organisatorisch abgebildet werden. Die Einrichtung eines Open-Source-Program-Offices (OSPO) hats sich mittlerweile als »best practice« etabliert.

Ein OSPO ist ein zentrales Team, das sich im Sinne eines ganzheitlichen Ansatzes um die unterschiedlichen Belange rund um Open-Source-Software innerhalb eines Unternehmens kümmert. Ursprünglich wurde dieser Ansatz hauptsächlich von IT-Firmen verwendet. Aufgrund der gestiegenen Bedeutung und Verbreitung von Open-Source-Software kann er jedoch durchaus auch für nicht-IT-Firmen sinnvoll und attraktiv sein, welche vor der Herausforderung stehen, Open-Source-Software im Unternehmen zu managen und entsprechende Prozesse und Werkzeuge einzuführen.

OSPO ist der international gebräuchlichste Begriff, oft wird eine ähnliche Organisationsform auch unter anderen Namen umgesetzt.

6.3.1 Aufgaben eines OSPOs

Zum Verantwortungsbereich eines OSPOs gehören typischerweise die folgenden Aufgaben:

- **Open-Source-Strategie**
  Hier geht um die Frage, wie Open-Source-Software – deren Verwendung, die Beteiligung an, aber auch das Initiieren neuer Open-Source-Projekte – die Unternehmens- und Produktstrategien unterstützen sowie helfen kann, die Unternehmensziele zu erreichen. Neben der Erstellung gehört auch die interne (und gegebenenfalls externe) Kommunikation der Open-Source-Strategie in Zusammenarbeit mit den entsprechenden Stakeholdern zu den Aufgaben des OSPOs.

- **Open-Source-Policy**
  Für den Umgang mit Open Source gilt es, Regeln festzulegen und innerhalb des Unternehmens zu verankern. Dies betrifft insbesondere die Verwendung von Open-Source-Software innerhalb der eigenen Organisation oder den eigenen Produkten, die Beteiligung an Open-Source-Projekten, aber auch das Starten eigener Projekte. Es ist Aufgabe des OSPOs, die Open-Source-Policy für das Unternehmen zu erstellen, zu kommunizieren und entsprechende Schulungen für die Mitarbeitenden bereitzustellen.

- **Open-Source-Prozesse**
  Die Definition, Implementierung, Durchführung und kontinuierliche Weiterentwicklung von Prozessen zum Management von Open-Source-Software im Unternehmen gehört ebenso zu den Aufgaben eines OSPOs. Zum Beispiel Prozesse, die helfen, Lizenzkonformität sicherzustellen.

- **Tools**
  Aufgrund der schieren Masse von Open-Source-Software in Unternehmen sind die damit verbundenen Prozesse ohne Automatisierung und Tool-Unterstützung insbesondere in größeren Unternehmen nicht mehr effizient durchzuführen. Aufgabe eines OSPOs ist es, entsprechende Werkzeuge für eine weitestgehende Automatisierung der Open Source-Management-Prozesse einzuführen, zu administrieren und möglicherweise auch selbst zu entwickeln.

- **Training**
  Der Umgang mit Open-Source-Software macht es erforderlich, dass die relevanten Mitarbeitenden entsprechendes Wissen aufbauen. Dies betrifft die Verwendung von Open-Source-Software unter Einhaltung der Compliance- und Security-Regeln, aber auch die Mitarbeit in Open-Source-Arbeitsgruppen sowie das Initiieren von Open-Source-Projekten und den damit verbundenen Aufbau von Communities. Das OSPO ist typischerweise dafür verantwortlich, ein entsprechendes Trainingsangebot für die Mitarbeitenden auszuarbeiten, bereitzustellen und kontinuierlich weiterzuentwickeln.

- **Kommunikation**
  Eine weitere Aufgabe eines OSPOs besteht darin, Transparenz über das Open-Source-Engagement eines Unternehmens herzustellen, sowohl intern für die eigenen Mitarbeitenden, als auch extern für die interessierte Öffentlichkeit, Entwickler-Communities, Partner und Kunden. Insbesondere wenn eigene Open-Source-Projekte gestartet werden, ist es wichtig, diese bekannt zu machen (beispielsweise über Blogposts, Vorträge auf Konferenzen oder Publikationen in den einschlägigen Medien). Ebenso ist es in der Regel gewünscht, Communities um diese Projekte herum aufzubauen und zu managen. Ein OSPO kann Entwicklungsteams bei diesen Aufgaben beraten und unterstützen.

- **Mitgliedschaften in Open-Source-Verbänden und Gremien**
  Im Open-Source-Bereich wird Einfluss typischerweise über Mitarbeit in Communities ausgeübt. Dies können projektspezifische Communities sein, aber auch größere Organisationen wie Open-Source-Foundations. Ein OSPO hat typischerweise die Aufgabe, Mitgliedschaften und Engagements in diesen Organisationen mit den strategischen Zielen, die ein Unternehmen mit Open Source verfolgt, in Einklang zu bringen. Solche Engagements müssen gemäß den strategischen Zielen des Unternehmens auf den Weg gebracht und unterstützt werden.

Die Fülle der Aufgaben eines OSPO ist also groß. Damit es nicht zu einem zentralen Flaschenhals wird, gilt es abzuwägen, welche Prozesse zentral über das OSPO laufen

müssen und welche Entscheidungen sinnvollerweise dezentral in den Entwicklungsabteilungen getroffen werden sollten. Eine wichtige Rolle spielen hierbei Tools zur Automatisierung und Unterstützung von Prozessen sowie Self-Services für die Mitarbeitenden.

6.3.2 Organisatorische Aspekte eines Open Source Program Offices

Es gibt keine allgemeingültige Blaupause für die Struktur und die organisatorische Verankerung eines OSPOs im Unternehmen. Vielmehr hängen diese von der Unternehmensstruktur, der Verwendung sowie der strategischen Bedeutung von Open-Source-Software für das Unternehmen ab. Oftmals fangen OSPOs klein an, mit nur wenigen Stellen, manchmal sogar nur einer Person.

Es gibt einerseits OSPOs, die komplett aus Vollzeitmitarbeitenden bestehen, andererseits gibt es welche, die komplett virtuell angelegt sind. In diesem Fall bestehen sie ausschließlich aus Teilzeitmitarbeitenden, welche die OSPO-Arbeit neben der Tätigkeit in ihrer eigentlichen Abteilung durchführen. Aufgrund der Vielfalt seiner Aufgaben muss ein OSPO mit vielen verschiedenen Unternehmensbereichen zusammenarbeiten, etwa mit:

- Entwicklungsabteilungen
- Produktmanagement
- Strategieabteilungen
- Lizenzmanagement
- Rechts-, IP-, Compliance- und Security-Abteilungen
- Schulungsabteilungen
- Unternehmenskommunikation und (technischem) Marketing
- interne IT-Abteilung
- Software-Beschaffung (Einkauf)
- Personalabteilung

Daher kann eine komplett virtuelle Organisation bestehend ausschließlich aus Teilzeitmitarbeitenden aus mehreren der oben genannten Abteilungen durchaus sinnvoll sein. Auch Mischformen sind denkbar. So könnte es ein Kernteam aus Vollzeitmitarbeitenden geben, die in einem virtuellen Ansatz von Teilzeitmitarbeitenden aus anderen Abteilungen unterstützt werden.

Ebenso vielfältig sind die Möglichkeiten zur Verankerung eines OSPOs im Unternehmen. Es kann einerseits ein zentrales OSPO geben, das für das gesamte Unternehmen zuständig ist. Besteht das Unternehmen jedoch aus Unternehmensteilen, die eher unabhängig voneinander agieren, so kann es durchaus ratsam sein, statt eines zentralen, dezentrale OSPOs in den einzelnen Unternehmensteilen aufzubauen.

Oft wird ein OSPO in der Berichtslinie des *Chief Technology Officers (CTO)* oder *Chief Operating Officers (COO)* angesiedelt.

Weiterführende Informationen zum Thema:

- ↗Creating an Open-Source-Program (↗TODO Group der Linux Foundation)
- ↗What does an open-source-program office do?
- ↗Does Your Organization Need an Open-Source-Program Office?

6.4 Open-Source-Foundations

Beim Engagement in Open-Source-Projekten, sei es als Contributor zu bestehenden Projekten oder beim Gründen von neuen Projekten, gibt es viele Aspekte zu berücksichtigen: Rechtliche Rahmenbedingungen, Modelle der Zusammenarbeit, erfolgreiche Communitypflege und mehr. Für eine einzelne Firma sind diese Fragen oft schwer zu beantworten, da die notwendige Expertise nicht immer in-house verfügbar ist und gerade zum Aufbau einer Community die Abhängigkeit von einer einzelnen Firma eine Hürde darstellt.

Eine Lösung stellen hier Open-Source-Foundations[43] dar. Das sind Organisationen, die als Zusammenschluss mehrerer Firmen und oft auch einzelnen Entwicklerinnen und Entwicklern, eine neutrale Basis für Zusammenarbeit an Open-Source-Projekten bieten. Die amerikanische ↗Linux Foundation und die europäische ↗Eclipse Foundation sind Beispiele für große, stark industrieorientierte Organisationen. Die ↗Apache Software Foundation ist ein Beispiel für eine eher community-orientierte Organisation. Es gibt aber auch eine Vielzahl weiterer Organisationen[44], insbesondere auch sogenannte »User-Led Foundations«[45], die von Firmen getrieben werden, die primär als Nutzer und nicht als Produzenten von Software agieren, oder industrie-spezifische Foundations, wie die ↗Academy Software Foundation für die Filmindustrie und ↗Eclipse Automotive für die Automobilindustrie.

Eine wichtige Funktion von Foundations ist, dass sie ein offenes Zusammenarbeitsmodell zur Verfügung stellen, oft als »Open Governance« bezeichnet.

Eine wichtige Funktion von Foundations ist, dass sie ein offenes Zusammenarbeitsmodell zur Verfügung stellen, oft als »Open Governance« bezeichnet. Dies ist notwendig, da Open-Source-Lizenzen zwar Software-Nutzenden weitgehende Freiheiten garantieren, aber nichts darüber aussagen, wie diese Software erstellt wird. Für ein tragfähiges Projekt ist es wichtig, Transparenz über Entscheidungsprozesse herzustellen, einen klaren Weg zur Beteiligung zu definieren, der eine auf viele Schultern verteilte Entwicklung fördert und eine neutrale Stelle zu haben, die zentrale Aspekte trägt, wie zum Beispiel Marken oder Projekt-Infrastruktur. Dies wird durch ein Open-Governance-Modell in einer Foundation erreicht.

Außerdem definieren Foundations rechtliche Rahmenbedingungen, die die Zusammenarbeit zwischen Unternehmen ermöglichen, ohne dass dafür spezifische neue Strukturen verhandelt und aufgebaut werden müssten. Dies deckt zum Beispiel auch kartellrechtliche Anforderungen ab.

43 Der Begriff »Foundations« hat sich als verallgemeinernder Begriff in diesem Kontext eingebürgert. Dahinter stecken verschiedene Formen von Organisationen, die ohne Gewinnerzielungsabsicht arbeiten. In Deutschland ist das oft der Verein. In Europa ist das häufig auch die belgische AISBL. In Amerika sind sogenannte 501(c)-Organisationen üblich.

44 Einen guten Einblick gibt das Community-gepflegte Foundation Directory. Es erhebt aber keinen Anspruch auf Vollständigkeit. https://flossfoundations.org/foundation-directory/

45 Siehe dazu zum Beispiel https://oss.cs.fau.de/2020/03/12/research-paper-the-ecosystem-of-openkonsequenz-a-user-led-open-Source-foundation-oss-2020/

Schließlich bieten Foundations eine wertvolle Ressource zum Austausch, zum Lernen oder der Vertretung gemeinsamer Interessen. Da die Herausforderungen beim Umgang mit Open-Source-Software für viele Firmen sehr ähnlich sind, bietet sich hier ein gemeinsamer Ansatz besonders an. Insofern kann die gezielte Beteiligung an Foundations ein wichtiger Teil der Open-Source-Strategie sein.

6.5 InnerSource

Mit »InnerSource« wird das Vorgehen beschrieben, Open-Source-Prinzipien auf die Software-Entwicklung innerhalb eines Unternehmens anzuwenden, ohne den Programmcode außerhalb des Unternehmens zu veröffentlichen.

Damit kann von den Vorteilen von Open-Source-Entwicklungsmodellen profitiert werden, ohne die Entwicklung öffentlich zu machen und ohne das Geschäftsmodell des Verkaufs von Software-Lizenzen aufzugeben. Dazu gehört, dass Quellcode innerhalb des Unternehmens über Team- und Projektgrenzen hinweg verfügbar gemacht wird und auch Beiträge von Personen, die nicht Mitglied des eigentlichen Teams sind, akzeptiert werden.

InnerSource kann so zu einer verbesserten Zusammenarbeit und zur Erhöhung von Entwicklungsgeschwindigkeit und Qualität führen sowie dazu beitragen, Kosten zu senken. Mehrfachentwicklungen können vermieden werden und Teams, die von Projekten abhängen, die nicht genug Entwicklungskapazitäten haben, können benötigte Features selbst beitragen. Dies gelingt insbesondere in Situationen, in denen verschiedene Unternehmensteile Software-Komponenten mit ähnlichen Anforderungen benötigen, zum Beispiel bei internen Standardbibliotheken, Infrastruktur-Komponenten oder Entwicklungs-, Test- und Deployment-Werkzeugen.

Es ist wichtig festzuhalten, dass InnerSource nicht gleichzusetzen ist mit Open Source. InnerSource-Code ist weiterhin proprietärer Code und die Dynamik eines InnerSource-Projekts wird anders sein als die eines öffentlich, in vielen Fällen durch Freiwillige, geführten Projektes. Dennoch lassen sich viele Konzepte übertragen und damit auch positive kulturelle Änderungen erwirken.

Es gibt eine Vielzahl von Beispielen für InnerSource-Initiativen. Das Buch ↗Adopting InnerSource, Principles and Case Studies beschreibt in einer Reihe von Fallstudien die erfolgreiche Anwendung von InnerSource in Unternehmen wie Bosch, Ericsson oder Paypal. Mehr Material findet sich in der ↗InnerSource Commons Community. Darunter auch eine Reihe von ↗InnerSource Patterns, in denen Taktiken zum Einsatz von InnerSource erfasst werden.

7 Open-Source-Compliance

›Compliance‹ ist ein englisches Wort. Der Duden von 2009 kennt es noch nicht.[46] Seine aktuelle Onlineversion sagt, die Wirtschaft verwende es in der Bedeutung »regelgerechtes, vorschriftsgemäßes, ethisch korrektes Verhalten«[47]. Und Wikipedia konstatiert, dass »Compliance [...] die betriebswirtschaftliche und rechtswissenschaftliche Umschreibung für die Regeltreue (auch Regelkonformität) von Unternehmen [sei], also die Einhaltung von Gesetzen, Richtlinien und freiwilligen Kodizes [meine]«[48], während man »im rechtlichen Bereich [...] mit dem Begriff Compliance grundsätzlich die Einhaltung von Regeln in Form von Recht und Gesetz (beschreibe)«[49].

Compliance meint mithin eine Aktivität: Es geht um die willentliche und planvolle Einhaltung von Vorgaben, nicht um eine gewissermaßen ›passive‹ Gesetzestreue. Das mag ein kleiner Unterschied sein, bei *Open-Source-Software* ist er entscheidend: Personen dürfen diese Art der Software in der Regel nicht nutzen, ohne vorher aktiv etwas getan zu haben.

Verknüpft werden Nutzungserlaubnis und Aktion über die Open-Source-Lizenzen. Diese haben eine typische Struktur: Zunächst werden den Nutzerinnen und Nutzern der Software Verwertungsrechte zugesprochen. Danach wird diese Erlaubnis jeweils an Bedingungen geknüpft. Verdeutlichen kann man sich das am Beispiel der einfachsten Open-Source-Lizenz, der MIT-Lizenz[50]. Sie beginnt mit einer *Copyright-Line* und der Aussage:

»Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the »Software«), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, [...]«

Und mit dem Nachsatz **»subject to the following conditions«** bindet sie diese Erlaubnis an die Bedingung:

»The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.«

Dies bedeutet tatsächlich, dass MIT-lizenzierte Software nur dann weitergegeben werden darf, wenn alle »substantial portions of the Software« mit dem jeweiligen Lizenztext gebündelt werden. Was ein wesentlicher Teil der Software ist und was nicht, ist letztlich eine Interpretationsfrage. Das aber heißt nicht, dass man die Lizenz ignorieren oder diesen Spielraum missinterpretieren darf. Eine gute Grundregel für Nutzerinnen und Nutzer ist es, den Lizenztext so wörtlich und so ernst wie möglich zu

Eine gute Grundregel für Nutzerinnen und Nutzer ist es, den Lizenztext so wörtlich und so ernst wie möglich zu nehmen.

46 vgl. Duden. Die deutsche Rechtschreibung, 25., völlig neu bearbeitete Aufl.; hrsg. v. d. Dudenredaktion, Mannheim, Wien u. Zürich, 2009 (Duden Band 1), S. 317 u. S. 635

47 vgl. https://www.duden.de/rechtschreibung/Compliance

48 vgl. https://de.wikipedia.org/wiki/Compliance_(BWL)

49 vgl. https://de.wikipedia.org/wiki/Compliance_(Recht)

50 vgl. https://opensource.org/licenses/MIT. Wir zitieren hier das Lizenztemplate, wie es die OSI angibt. Die MIT lizenzierte Software selbst verwendet immer eine instanziierte Lizenz, will sagen: eine Lizenz mit konkretisierter Copyright-Line. Und wenn die Lizenz im Folgenden fordert, den Lizenztext und die voranstehende Copyright-Line der Software beizulegen, dann ist damit eben diese konkretisierte Copyright-Line gemeint. Es reicht also nicht aus, das Template dem weitergegebenen Bundle hinzuzufügen. Man muss zur Erfüllung der Lizenz immer nach dem wirklichen Lizenztext im Repository suchen.

nehmen. Allerdings bedeutet dies auch, dass nichts getan werden muss, wenn die MIT-lizenzierte Software nicht weitergeben, also nur für eigene Zwecke auf eigenen Rechnern genutzt wird.

Damit sind **typische Merkmale einer Open-Source-Lizenz** definiert:

- die Zuschreibung der Nutzungsrechte
- die Formulierung der zu erfüllenden Bedingungen
- die Verknüpfung der Bedingungen an Nutzungsszenarien
- und die Existenz von Interpretationsspielraum, den Sprache immer eröffnet.

Nun werden wir weitere Lizenzen diskutieren, allerdings nicht in allen Einzelheiten. Denn es gilt ganz allgemein: Zuletzt erfordert die angemessene Erfüllung der Lizenzbedingungen immer den eigenen Blick in den Lizenztext. Sekundäre Erläuterungen – wie diese – sind gut für das Verständnis, für die rechtliche Einschätzung sind sie unerheblich, selbst wenn sie Anspruch erheben, eine Anleitung zu sein.

> Das Recht, Open-Source-Software zu nutzen, kann in der Tat nicht direkt mit Geld erworben werden.

Aufmerksame Leserinnen und Leser werden sich schon gefragt haben, wo in solchen Lizenzen von Gebühren die Rede ist, haben sie doch selbst unter Umständen schon einmal für Open-Source-Software bezahlt – und das, obwohl man Open-Source-Software doch angeblich nicht kaufen könne. Tatsächlich liegt darin kein Widerspruch: Das Recht, Open-Source-Software zu nutzen, kann in der Tat nicht direkt mit Geld erworben werden.[51] Stattdessen ›erwirbt‹ man die Nutzungsrechte, indem man tut, was die Lizenzen fordern. Anders gesagt: Open-Source-Software folgt dem Prinzip *Paying by Doing*.[52] Wer also trotzdem Geld für Open-Source-Software ausgegeben hat, hat für einen Service bezahlt, der mit und für Open-Source-Software erbracht wird[53], nicht aber für das Recht, die Software zu einem beliebigen Zweck zu nutzen, sie zu untersuchen, sie an andere weiterzugeben und sie zu verändern.[54]

Damit ist der Rahmen aufgespannt, den es zu füllen gilt, wenn von Open-Source-Compliance die Rede ist, nämlich

- über ↗ grundsätzliche Herausforderungen
- über ↗ Tools zur Erfüllung
- über ↗ Randaspekte der Erfüllbarkeit und
- über ↗ rechtliche Basiskonstrukte.

51 vgl. die Open-Source-Definition ↗ Kap. 2.2
52 vgl.: Reincke and Sharpe: Open Source License Compendium, Release 1.0.2, 2018, S. 103 https://telekom.github. io/oslic/releases/oslic.pdf
53 vgl. die Open-Source-Geschäftsmodelle ↗ Kap. 5
54 vgl. https://www.gnu.org/philosophy/free-sw.html#four-freedoms

7.1 Open-Source-Compliance als Aufgabe

Software fällt in aller Regel unter das Urheberrecht[55]: Jeder Softwareentwickler hat per Gesetz zunächst einmal das Urheberrecht an dem, was er programmiert. Oft tritt er als Urheber den größten Teil dieser Rechte[56], nämlich die sogenannte Verwertungsrechte, per Arbeits- oder Liefervertrag an seine Firma oder Kundinnen und Kunden ab. Die Ausgangslage ändert sich dadurch nicht wesentlich: Software wird zu Open-Source-Software, indem der Inhabende der Verwertungsrechte – sei es der Programmierer, sei es der nachgelagerte Verwerter – die Software unter einer Open-Source-Lizenz[57] veröffentlicht. Eine Lizenz ihrerseits wird zu einer Open-Source-Lizenz, indem die Open-Source-Initiative[58] – gern als OSI abgekürzt – die Lizenz in die Liste der offiziellen Open-Source-Lizenzen aufnimmt[59]. Und dafür wiederum muss die Lizenz den 10 Kriterien der *Open-Source-Definition* (OSD) genügen[60], die zu erfüllen, die *OSI* den Open-Source-Lizenzen abverlangt. So einfach ist das – im Prinzip.

Der Gewinn, den die *OSI* mit der *OSD* in das Open-Source-Spiel bringt, liegt darin, dass sie anhand der OSD nur solche Lizenzen als Open-Source-Lizenz klassifiziert, die den Nutzerinnen und Nutzern der Software die entsprechenden Rechte auch tatsächlich gewähren. Wer Software nutzt, die unter einer bestätigten Open-Source-Lizenz veröffentlicht worden ist, darf also sicher sein, dass diese den Nutzern tatsächlich die Rechte gewährt, die sie zu deren Nutzung benötigen. Es ist nicht mehr nötig, bei jedem Softwarepaket, das man verwenden möchte, juristisch aufwendig prüfen zu lassen, dass man die Nutzungsrechte auch tatsächlich hat. Das macht die Verwendung von Open-Source-Software einfach.[61]

Wer Software nutzt, die unter einer bestätigten Open-Source-Lizenz veröffentlicht worden ist, darf also sicher sein, dass diese den Nutzern tatsächlich die Rechte gewährt, die sie zu deren Nutzung benötigen.

55 Als eine erste Annäherungsmöglichkeit zu diesem Thema sei auf die beiden Wikipedia-Artikel *Softwarerecht* (https://de.wikipedia.org/wiki/Softwarerecht_(Deutschland) und Urheberrecht (https://de.wikipedia.org/wiki/Urheberrecht) verwiesen.

56 Das Recht, als Autor genannt zu werden, kann nicht abgetreten werden. Oder anders gesagt: Immer noch darf niemand anderes als der Programmierer selbst sich als Autor des Codes bezeichnen.

57 Im Kern gibt es eine öffentliche Liste der offiziellen Open-Source-Lizenzen, die von der Open-Source-Initiative verwaltet wird: https://opensource.org/licenses/alphabetical

58 vgl. https://opensource.org/about

59 Um eine Lizenz berechtigt aufzunehmen, hat sich die OSI einen *Approval Process* gegeben. https://opensource. org/approval

60 vgl. https://opensource.org/osd Die *Open-Source-Definition* ist also nur mittelbar eine Definition von *Open-Source-Software*. Tatsächlich enthält sie Kriterien, die die Open-Source-Lizenzen erfüllen müssen, nicht die Software. Aber diese Kriterien beziehen sich auf Rechte, die die Lizenzen den Softwarenutzerinnen und -nutzern einräumen müssen, wenn sie Open-Source-Lizenzen sein wollen. Das bedeutet umgekehrt, dass die Nutzerinnen und Nutzer einer unter einer offiziellen Open-Source-Lizenz vertriebenen Software immer schon wissen, dass sie bestimmte Rechte haben. Über ihre Pflichten wird dabei jedoch nichts gesagt!

61 Formal gesehen bleibt es leider doch eine Aufgabe eines jeden Nutzers sich dieser Rechteübertragung persönlich zu vergewissern. Denn juristisch geklärt ist nur das, was vor Gericht entschieden ist. Und ob die OSD stimmt, ist unseres Wissens nach noch nicht Thema vor Gericht gewesen. Allerdings: Die Tatsache, dass so wenig vor Gericht verhandelt worden ist, ist auch ein Zeichen dafür, wie wenig strittig das Thema ist. Mit anderen Worten: Wer sich die OSD in Sachen Rechtezuschreibung zu eigen macht und sich sein Leben damit vereinfacht, ist in guter Gesellschaft.

Dennoch ist Open-Source-Compliance eine komplexe Sache. Denn die lizenzkonforme Nutzung von Open-Source-Software durch aktive Erfüllung der entsprechenden Open-Source-Lizenzbedingungen muss verschiedene Dimensionen berücksichtigen:

- Die Open-Source-Definition der OSI legt nur fest, welche Rechte die Lizenzen den Nutzerinnen und Nutzern einer so lizenzierten Software einräumen müssen. Dass diese Nutzungsrechte kostenlos weitergegeben werden müssen, ergibt sich aus der Pflicht, den Sourcecode zu nicht mehr Kosten zur Verfügung zu stellen, als man vernünftigerweise für eine (elektronische) Softwarekopie verlangen darf und den einzuräumenden Rechten der Bearbeitung und der Weitergabe dieses Codes[62]. Darüber hinaus überlassen es die *OSD* und die *OSI* aber den einzelnen Lizenzen, den Nutzerinnen und Nutzern der Software Pflichten aufzuerlegen – sofern und soweit diese Pflichten die zu gewährenden Rechte nicht eingrenzen. Die zu gewährenden Rechte sind normiert, die Pflichten nicht. Darum muss zuletzt immer anhand der einzelnen Lizenz ermittelt werden, welche Pflichten genau erfüllt werden müssen. Will sagen: Was das *Paying-by-Doing*-Prinzip den Nutzerinnen und Nutzern im Einzelfall aufbürdet.

- Nicht alle Lizenzen, die wie eine Open-Source-Lizenz aussehen oder sich dafür ausgeben, sind von der OSI als solche bestätigt worden. Manche von ihnen erfüllen dennoch alle Kriterien der OSD und werden möglicherweise noch zu einem späteren Zeitpunkt akzeptiert. Andere stimmen fast wortgleich mit einer bereits akzeptierten Open-Source-Lizenz überein und werden doch einer mehr oder minder kleinen Abweichung wegen niemals offiziell als Open-Source-Lizenz anerkannt werden.[63] Außerdem gibt es Lizenzen, deren ›Open-Source-Haftigkeit‹ strittig geblieben ist.[64] Und schließlich begegnet man Lizenzen, die Open-Source-Lizenzen sein wollen, inhaltlich aber auf eine proprietäre Lizenz hinauslaufen. Darum wird auch unterschieden zwischen *echten Open-Source-Lizenzen, potentiellen OpenSource-Lizenzen, unechten* resp. *strittigen Open-Source-Lizenzen* und *Fake-Open-Source-Lizenzen*. Trotzdem folgt auch bei diesen ›Nebenformen‹ direkt aus dem Urheberrecht, dass Personen, die eine so lizenzierte Software nutzen, unbedingt die in der Lizenz genannten Forderungen erfüllen müssen: Es ist das Recht eines jeden Urhebers, sein Werk mit welchen Nutzungsbedingungen auch immer zu verknüpfen. Die Erkenntnis, dass eine Lizenz keine (echte) Open-Source-Lizenz ist, enthebt also nicht davon, ihre Vorgaben zu berücksichtigen.

Es ist das Recht eines jeden Urhebers, sein Werk mit welchen Nutzungsbedingungen auch immer zu verknüpfen.

62 vgl. https://opensource.org/osd Kriterium Nr. 2. Source Code, Nr. 1. Free Redistribution, Nr. 3. Derived Works

63 So stimmt die JSON Lizenz (vgl. https://www.json.org/license.html) mit Ausnahme eines Satzes mit der MIT-Lizenz (vgl. https://opensource.org/licenses/MIT) überein, einer der ältesten und kürzesten offiziellen Open-Source-Lizenzen: Dieser Satz, den die JSON Lizenz mehr hat, lautet »The Software shall be used for Good, not Evil.« So sehr wir mit dieser Bedingung auch sympathisieren mögen, so sehr verstößt sie doch gegen die OSD-Klauseln 5 und 6 (vgl. https://opensource.org/osd)

64 Solch eine ›strittige‹ Lizenz ist z. B. die *Server Side Public License* (SSPL), unter der die MongoDB zuletzt relizenziert worden ist. Initial war die MongoDB unter der AGPL-3.0 distribuiert worden, einer anerkannten Open-Source-Lizenz (vgl. https://opensource.org/licenses/AGPL-3.0). Die Firma hinter der MongoDB empfand jedoch, dass immer mehr MongoDB-as-a-Service-Angebote in der Cloud-Welt entstanden, ohne dass die AGPL wirklich berücksichtigt würde. Zuerst wollte sie die Bedeutung der AGPL durch eine Klausel klarstellen. Dann schuf sie ihre eigene Lizenz, eben die SSPL, die mit Ausnahme eines Paragraphen wortgleich mit der AGPL übereinstimmt. Der ›Approval‹-Prozess bei der OSI gestaltete sich ›komplex‹, sodass die Firma den Antrag schließlich zurückzog. (vgl. dazu https://en.wikipedia.org/wiki/Server_Side_Public_License). Trotzdem ähnelt die SSPL der AGPL weiterhin – schon rein textlich. Und vom Urheberrecht her ist es – wie gleich noch ausgeführt – schlicht egal, ob eine Lizenz eine echte, eine potentielle, eine unechte oder eine Fake-Open-Source-Lizenz ist: Nutzt man die so lizenzierte Software, hat sie deren Bedingungen zu erfüllen

- Obwohl die Lizenzen von der Art dessen, was sie den Nutzerinnen und Nutzern einer so lizenzierten Software als Gebot oder Verbot auferlegen, gut klassifizierbar sind – dem widmet sich der folgende Abschnitt –, lässt sich das nur auf einer abstrakteren Ebene zusammenfassen. Letztlich unterscheiden sich die Forderungen der Lizenzen im Detail so, dass jeder Nutzer für jeden Einzelfall feststellen muss, was genau er tun (und unterlassen) muss. Klassifikationen und Gruppierungen, wie im Folgenden formuliert, sind hilfreich. Sie als alleinigen Maßstab zu nehmen und den eigentlichen Lizenztext außen vor zu lassen, ist unangebracht.

> Zu einer adäquaten Open-Source-Compliance-Analyse gehört mithin immer auch eine Begutachtung des Einsatzszenarios.

- Eine nächste Komplikation entsteht dadurch, dass die Wirkung von Open-Source-Lizenzen vom Kontext abhängt: Wann welche ihrer Bedingungen erfüllt werden müssen, hängt unter anderem von der Art der Verwendung ab, die die Forderung nach Erfüllung erst anstößt. Ein weit verbreiteter Trigger ist etwa die »Weitergabe« der Open-Source-Software, wobei schon das genaue (juristische) Verständnis, was Weitergabe im Rechtssinne ist, von Land zu Land unterschiedlich sein kann. Andere Trigger sind die Möglichkeit, über das Netz auf die Software zugreifen zu können oder die Tatsache, dass die Software modifiziert worden ist. Zu einer adäquaten Open-Source-Compliance-Analyse gehört mithin immer auch eine Begutachtung des Einsatzszenarios und ob die dadurch ausgelösten Bedingungen mit dem geschäftlichen Zweck des Einsatzes in Einklang zu bringen sind.

- Eine letzte Steigerung der Komplexität ergibt sich aus der Tatsache, dass das, was man tun muss, um die Open-Source-Software lizenzkonform zu nutzen, auch von der Position in einer Softwarearchitektur abhängt, an der die fragliche Open-Source-Komponente verwendet werden soll: Ein Modul oder eine Bibliothek in das eigene Werk einzubetten, kann einen starken Copyleft-Effekt auslösen; eine Open-Source-Software als eigenständige Komponente mit eigenem Adressraum zu verwenden, eher nicht. Der Versuch, solche komplexen Zusammenhänge mit verführerisch vereinfachenden Regeln zu managen – etwa mit der Vorgabe, dann doch besser überhaupt keine Open-Source-Software mit (starkem) Copyleft-Effekt zu verwenden – grenzen den Spielraum von Nutzerinnen und Nutzern überstark ein und bringen sie ihren Mitbewerbern gegenüber unnötig ins Hintertreffen. Besser ist es, bei der Open-Source-Compliance-Analyse auch die Softwarearchitektur zu dokumentieren und den Verwendungskontext innerhalb dieser Architektur zu berücksichtigen.

Der Weg, wie Software zu Open-Source-Software wird, ist einfach. Der Weg, sie lizenzkonform zu nutzen, wird gelegentlich steinig. So erklären manche Managerinnen und Manager Open-Source-Compliance zu einem bloßen Risiko, das doch gemindert oder auch einfach nur so übernommen werden könne, weil man sowieso nicht ›erwischt‹ werde und – wenn doch einmal – pekuniär nicht so hoch ›bestraft‹ werde. Diese Sichtweise ist irrig! Legal kann man Open-Source-Software nur unter Einhaltung der Lizenzbedingung nutzen – oder eben gar nicht. Jedes Unternehmen, dessen Code-Of-Conduct seine Mitarbeiterinnen zur Einhaltung der Gesetze verpflichtet, verlangt von ihnen damit auch, Open-Source-Software nur lizenzkonform zu nutzen – egal, wie komplex die Bedingungen sind.

> Legal kann man Open-Source-Software nur unter Einhaltung der Lizenzbedingung nutzen – oder eben gar nicht.

Es gibt aber auch eine gute Nachricht: Zuletzt wird es doch wieder einfach. Wer Open-Source-Compliance für ein Produkt herstellen will, braucht nämlich ›nur‹

- die zugehörige Bill-Of-Material (BOM) zu generieren, indem
  - eine Liste aller verwendeten Open-Source-Komponenten des produktspezifischen Softwarestacks erstellt und
  - zu jedem Eintrag darin die Homepage, die Versionsnummer, das Softwarerepository und die Lizenz vermerkt beziehungsweise verlinkt und notiert wird, ob es sich um eine App beziehungsweise ein Programm oder ein Modul beziehungsweise eine Bibliothek handelt,[65]
- die Softwarearchitektur zu dokumentieren,[66]
- sich auf der Basis dieser Informationen von Compliance-Kompetenzträgerinnen und -trägern eine Liste dessen erstellen zu lassen, was zu tun ist, um die Software in dem Produkt lizenzkonform zu verwenden[67] und
- diese Aufgabenliste abzuarbeiten.

> Systemisch gesehen, wird es zuletzt doch wieder einfach, Open-Source-Compliance für ein Produkt herzustellen.

65 Die Arbeit, solche Informationen zusammenzutragen, ist die, die schon am besten und meisten im Sinne einer Automatisierung durch Open-Source-Tools unterstützt wird. (vgl. https://oss-compliance-tooling.org/ Tooling-Landscape/OSS-Based-License-Compliance-Tools/)

66 Ein besonders gutes Tool, um Softwarearchitekturen in Diagrammform zu dokumentieren, ist draw.io, gehostet unter https://diagrams.net/, im Quellcode veröffentlicht unter https://github.com/jgraph/drawio und veröffentlicht unter der Apache-2.0 Lizenz https://github.com/jgraph/drawio/blob/dev/LICENSE

67 Die toolgestützte automatisierte Anwendung von Compliance-Kompetenz ist das z. Zt. noch am wenigsten beackerte Feld. Ein Ansatz in sehr frühem Stadium ist OSCake, https://github.com/Open-Source-Compliance/OSCake. Dies zeigt, dass wir bei der Open-Source-Compliance und der Erstellung von vorgegeben Compliance-Artefakten – trotz aller Anstrengungen – immer noch sehr viel ›Handarbeit‹ vor uns haben

7.2 Lizenztypen und Compliance-Aktivitäten

Bevor wir Ihnen eine Klassifikation für die Open-Source-Lizenzen anbieten und die mit ihnen typischerweise verknüpften Bedingungen darstellen, wollen wir Ihnen das Abgrenzungskriterium erläutern, anhand dessen die Kategorien gebildet werden. Denn in der Tat leiten sich die üblichen Klassen

- *permissive Open-Source-Lizenzen* ohne Copyleft-Effekt
- *Open-Source-Lizenzen mit schwachem Copyleft-Effekt*
- *Open-Source-Lizenzen mit starkem Copyleft-Effekt*

von der Art ab, wie der *Copyleft-Effekt* darin eingebettet ist. Insofern ist er es wert, auch gesondert erläutert zu werden:

»Copyleft« ist ein Wortspiel: Das Copyright diene der Wahrung der Rechte der Urheber. Das Copyleft dagegen möchte einmal gewährte Rechte als Allgemeingut erhalten.

7.2.1 Der Copyleft-Effekt (als Abgrenzungskriterium)

Erfunden wurde der Begriff »Copyleft« von Richard Stallman als »Wortspiel«.[68] Seiner Meinung nach nutzen Copyright-Owner das Copy**right**, um den Nutzerinnen und Nutzern von Software Rechte vorzuenthalten, die ihnen gehören sollten. Demgegenüber solle das Copy**left** sicherstellen, dass diese Rechte den Nutzerinnen und Nutzern nicht genommen werden können. Copy**left** bezeichne deshalb eine Methode, »[...] ein Programm (oder anderes Werk) frei zu machen und zu verlangen, dass alle modifizierten und erweiterten Programmversionen ebenfalls frei sind«. Der Urheber lizenziert seinen Code so, dass nicht nur der eigene Code frei genutzt, frei geändert und frei weitergegeben werden darf, sondern dass auch alle **Änderungen, Ergänzungen und Ableitungen** im gleichen Sinn **frei genutzt werden dürfen**, die andere Entwicklerinnen und Entwickler in den Entwicklungsprozess einbringen.[69] Verkürzt gesagt, meint »Copyleft« also, dass man seine Bearbeitungen nur unter denselben Bedingungen weitergeben darf, unter denen man den Ausgangsstand seiner Bearbeitungen erhalten hat.

In der Praxis ergibt sich außerdem die Notwendigkeit, zwischen **starkem** und **schwachem Copyleft** zu unterscheiden:

Das *starke Copyleft* will dafür sorgen, dass die Software, die ein so lizenziertes Werk als eine konstitutive Komponente verwendet, unter denselben Bedingungen weitergegeben wird, unter denen man die selbst eingebettete Komponente erhalten hat.

68 vgl. https://de.wikipedia.org/wiki/Copyleft
69 vgl. GNU-Projekt http://www.gnu.org/copyleft/copyleft.de.html

Die ursprünglichen Lizenzbedingungen erstrecken sich also nicht nur auf Änderungen an oder Ergänzungen zu der verwendeten Komponente selbst, sondern auch auf den ›eigenen‹ Code, der diese Komponente verwendet.[70]

Das *schwache Copyleft* möchte dies nur für das übernommene Werk und dessen direkte Änderungen sicherstellen. So wird es möglich, zu einer unter schwachem Copyleft stehenden Bibliothek im Rahmen eines abgeleiteten Werkes hinzugefügten, unabhängigen Code unter eine andere Lizenz zu stellen. Für diese andere Lizenz gelten weniger oder keine Einschränkungen. Nur bei der eingebetteten Komponente, die unter einer Lizenz mit schwachem Copyleft-Effekt distribuiert wird, kommen diese Regelungen zur Geltung.[71]

7.2.2 Open-Source-Lizenz-Typologie

Damit können die bekannten Lizenzen in Gruppen eingeordnet werden:[72]

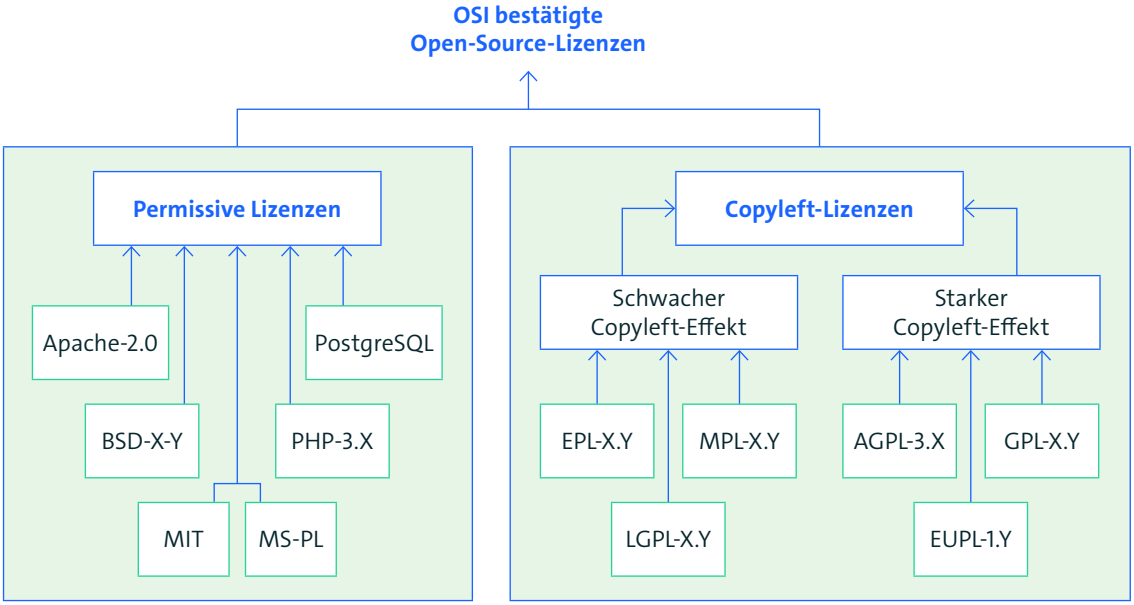

Abb. 1: Open-Source-License Classification (Auszug)

70 So kann das *starke Copyleft* auch Lizenzkonflikte verursachen. Denn setzte ein Programm zwei unterschiedlich lizenzierte Bibliotheken voraus, deren Lizenzen jeweils ein starkes Copyleft proklamierten, dann könnte das Programm nicht mehr lizenzkonform weitergeben werden – eben weil beide eingebetteten Komponenten forderten, dass der übergeordnete Teil jeweils unter derselben Lizenz distribuiert werden müsse, wie die Komponente.

71 Die *OSI* diskutiert, den Begriff *Copyleft* überhaupt aufzugeben und durch *reciprocal licensing* zu ersetzen. Die Wörter weak und strong wären dann durch die Beschreibung des »reciprocity scopes« zu ersetzen. (vgl. https://opensource.org/node/875). Dass sich das starke und das schwache Copyleft anhand ihrer Wirkung, ihres Focus unterschieden, haben wir ausgeführt. Den Begriff ›Copyleft‹ nicht mehr zu erwähnen, ist unmöglich. Es gibt im Kontext der Freien Open-Source-Software kaum einen stärker etablierten Terminus.

72 vgl.: Reincke and Sharpe: Open Source License Compendium, Release 1.0.2, 2018, S. 23 (= https://telekom.git-hub.io/oslic/releases/oslic.pdf). Neben dieser gibt es eine zweite Art der Gruppierung, bei der man die Open-Source-Lizenzen hinsichtlich ihres Umgangs mit Softwarepatenten analysiert. vgl. dies. a. a. O., Chapter 3.1 »The problem of implicitly releasing patents«.

Permissive Lizenzen bilden die erste Untergruppe der Open-Source-Lizenzen. Als OSI zertifizierte Lizenzen gewähren sie den Nutzerinnen und Nutzern alle Rechte, die in der OSD genannt sind, insbesondere das Recht, die Software zu nutzen, sie weiterzugeben, sie zu verändern und sie verändert weiterzugeben (sofern Zugriff auf den Quellcode vorliegt.)

Darüber hinaus haben permissive Lizenzen **keinen Copyleft-Effekt** – weder einen schwachen noch einen starken. Diese Lizenzen überlassen es den Bearbeitenden der Software, ob sie ihre eigene übergreifende Arbeit oder ihre Änderungen an dem übernommenen Werk auch wieder als freie Software weitergeben. **Sie stellen ihnen frei, das Ergebnis ihrer Bearbeitung zu proprietärer Software zu machen** – deshalb der Name dieser Kategorie: *Permissive Lizenzen* erlauben es den Bearbeitenden das Ergebnis ihrer Verbesserungen und Veränderungen sozusagen für sich zu behalten, obwohl diese Arbeit auf einer freien Softwarebasis aufsetzt.

Permissive Lizenzen lassen es zudem zu, dass die Bearbeitenden das Ergebnis ihrer Eingriffe als proprietäre Software in Binärform weitergeben können.

Und trotzdem legt auch jede permissive Lizenz den Nutzerinnen und Nutzern der so lizenzierten Software Pflichten auf, die sie selbst dann zu erfüllen haben, wenn sie ihr Werk als Ganzes in Binärform unter einer anderen Lizenz weitergeben.

- Alle permissiven Lizenzen wollen, dass Paketen oder Produkten, die eine so lizenzierte Software enthalten, der Lizenztext beigefügt wird.
- Einige permissive Lizenzen erwarten ferner, dass diesen Bundles noch spezielle Dateien beigelegt werden, sofern diese im Repository enthalten sind
- Einige permissive Lizenzen untersagen zudem, dass mit Marken, Namen oder Ähnlichem geworben wird.

Zur **Gruppe der permissiven Lizenzen** gehören mindestens:

- *Apache-2.0*, die Apache License, Version 2.0,[73]
- Berkeley Software Distribution Lizenzen,[74]
  - *BSD-2-Clause*,
  - *BSD-3-Clause*,
- *MIT*-Lizenz,
- *MS-PL* (Microsoft Public License),
- *PostgreSQL*-Lizenz,
- *PHP-3.0*-Lizenz.

> Permissive Lizenzen heißen »permissive Lizenzen«, weil sie dem Nutzer am wenigsten Pflichten auflisten.

73 Wir folgen bei der Lizenzbezeichnung (der kursive Teil) der SPDX-Nomenklatur http://spdx.org/licenses/, die u. U. das Ziel einer einfachen und effizienten Wiederauffindbarkeit von Lizenzen durch einheitliche Bezeichnungen und Identifier verfolgt.

74 Die MIT-Lizenz und die BSD-Lizenzen verlangen eine erhöhte Aufmerksamkeit, da sie auf die Idee eines Templates hin angelegt sind: Man übernimmt den Lizenztext, spezifiziert die Copyright-Line und nennt das Ganze XYZ-Lizenz. Bei den BSD-Lizenzen und der MIT-Lizenz ist es also notwendig, den Lizenztext selbst anzusehen, um festzustellen, ob es wirklich eine BSD-Lizenz resp. MIT-Lizenz ist.

Lizenzen mit schwachem Copyleft bilden die zweite Untergruppe der Open-Source-Lizenzen. Auch sie gewähren den Nutzerinnen und Nutzern als *OSI*-zertifizierte Lizenzen die *OSD*-Rechte, also insbesondere das Recht, die Software zu nutzen, sie weiterzugeben, sie zu verändern und sie verändert weiterzugeben. Dass die Nutzerinnen und Nutzer Zugriff auf den zur Veränderung notwendigen Quellcode haben, garantiert der in die Lizenz eingebettete schwache Copyleft-Effekt:

Diese Lizenzen fordern nämlich – nebst all den anderen Bedingungen, die sie auch erfüllt sehen wollen –, dass Veränderungen und Verbesserungen an der Software selbst unter denselben Bedingungen genutzt und weitergegeben werden wie die, unter denen die Bearbeiterin das Original erhalten hat, also unter derselben Lizenz. Diese Bedingungen schließen ein, dass die einmal erhaltene Software – verändert oder unverändert – auch Dritten gegenüber wieder im Quellcode zugänglich gemacht werden muss.[75]

Die Open-Source-Lizenzen mit schwachem Copyleft dehnen diese Verpflichtung aber nicht auf die sozusagen ›übergeordnete‹ Software aus, die eine so lizenzierte Software mit schwachem Copyleft als eingebettete Komponente nutzt.

Damit erlegen die Lizenzen mit schwachem Copyleft den Nutzerinnen und Nutzern in etwa folgende Pflichten auf:

Bei Lizenzen mit schwachem Copyleft betrifft der Copylefteffekt nur die so lizenzierte Software selbst.

- Auch alle Lizenzen mit schwachem Copyleft-Effekt wollen, dass Paketen oder Produkten, die eine so lizenzierte Software enthalten, der Lizenztext beigefügt wird.
- Außerdem erwarten alle Lizenzen mit schwachem Copyleft-Effekt dass der Quellcode der so lizenzierten Komponente auf die eine oder andere Weise zugänglich gemacht wird, unabhängig davon, ob die Software verändert worden ist oder nicht.
- Einige von den Lizenzen mit schwachem Copyleft-Effekt erwarten darüber hinaus, dass diesen Bundles noch spezielle Dateien beigelegt werden, sofern diese im Repository enthalten sind.
- Andere möchten, dass die Copyright-Angaben aller beteiligten Programmierenden auffällig (»conspicuously«) angezeigt werden, also nicht nur innerhalb des Quellcodes sichtbar sind.[76]
- Und schließlich untersagen auch einige dieser Lizenzen, dass mit Marken, Namen oder Ähnlichem geworben wird.

75 siehe auch ↗Copyleft-Effekt im Allgemeinen
76 vgl. https://opensource.org/licenses/LGPL-2.1, § 1

Zur Gruppe der Lizenzen mit schwachem Copyleft gehören mindestens:

- *EPL* (= Eclipse Public License 1.0/2.0),[77]
- *LGPL* (= GNU Lesser General Public License, Version 2.1/3.0),
- *MPL* (= Mozilla Public License, Version 1.1/2.0),
- *MS-RL* (= Microsoft Reciprocal License).

Lizenzen mit starkem Copyleft bilden die dritte Gruppe der Open-Source-Lizenzen. Auch sie gewähren den Nutzerinnen und Nutzern als *OSI* zertifizierte Lizenzen die *OSD*-Rechte, also insbesondere das Recht, die Software zu nutzen, sie weiterzugeben, sie zu verändern und sie verändert weiterzugeben. Dass die Nutzerin Zugriff auf den zur Veränderung notwendigen Quellcode haben, garantiert der in die Lizenz eingebettete starke Copyleft-Effekt:

Diese Lizenzen fordern – nebst all den anderen Bedingungen, die sie auch erfüllt sehen wollen – dass Veränderungen und Verbesserungen unter denselben Bedingungen genutzt und weitergegeben werden wie die, unter denen die Bearbeitenden das Original erhalten haben und zwar die Veränderungen und Verbesserungen der Originalsoftware an sich und die der übergreifenden ›System-Software‹, die das Original als Subkomponente benutzt. Diese Bedingungen schließen ein, dass die Veränderungen der erhaltenen Software ebenso im Quellcode zugänglich gemacht werden, wie das auf dem Original oder der veränderten Version aufsetzende neue Ganze.[78]

> Bei Lizenzen mit starkem Copyleft muss auch die Software, die so lizenzierte Bibliotheken / Module nutzen, unter derselben Lizenz verteilt werden, unter der die benutzten Bibliotheken / Module verteilt worden sind.

- Auch alle Lizenzen mit starkem Copyleft-Effekt wollen, dass Paketen oder Produkten, die eine so lizenzierte Software enthalten, der Lizenztext beigefügt wird.

- Außerdem erwarten alle Lizenzen mit starkem Copyleft-Effekt, dass der Quellcode der so lizenzierten Komponente selbst auf die eine oder andere Weise zugänglich gemacht wird, unabhängig davon, ob die Software verändert worden ist oder nicht.

- Zusätzlich erwarten alle Lizenzen mit starkem Copyleft-Effekt, dass auch der Code derjenigen Software unter derselben Lizenz zugänglich gemacht wird, der die eigentliche Software als Komponente (in demselben Adressraum = Library, Module) nutzt. Dies betrifft insbesondere die Software, die man selbst auf der Basis dieser Komponenten entwickelt: Sie ist dann nicht mehr frei in der Wahl ihrer eigenen Lizenz.

- Einige von den Lizenzen mit starkem Copyleft-Effekt erwarten darüber hinaus, dass diesen Bundles noch spezielle Dateien beigelegt werden, sofern diese im Repository enthalten sind.

- Andere möchten, dass die Copyright-Angaben aller beteiligten Programmierer auffällig (»conspicuously«) angezeigt werden, also nicht nur innerhalb des Quellcodes sichtbar sind.[79]

- Und schließlich untersagen auch einige dieser Lizenzen, dass mit Marken, Namen oder Ähnlichem geworben wird.

77 Die Einordnung der EPL ist umstritten, teilweise wird sie als Lizenz mit starkem Copyleft gesehen.
78 siehe auch Copyleft-Effekt im Allgemeinen
79 vgl. https://opensource.org/licenses/GPL-2.0, § 1

Zur Gruppe der Lizenzen mit starkem Copyleft gehören mindestens:

- *GPL* (= GNU General Public License, Version 2.0/3.0),
- *AGPL* (= GNU Affero General Public License, Version 3.0),
- *EUPL* (= European Union Public License, Version 1.0–1.2).[80]

7.2.3 Compliance-Verpflichtungen in der Übersicht

Die Feinheiten der Lizenzen dürfen von einem nicht ablenken:

Für sie alle gilt das *Paying-by-Doing*-Prinzip. Keine von ihnen bietet die Option, die Nutzungsrechte unter Aufhebung der Pflichten finanziell zu erwerben. Stattdessen erwartet selbst die ›permissivste‹ Lizenz von den Nutzerinnen und Nutzern einer so lizenzierten Software, dass sie unter bestimmten Umständen etwas tun. Um also Open-Source-Software lizenzkonform zu nutzen, reicht es nicht, in generalisierten Kategorien zu denken. Nur die konkrete Lizenz legt fest, was genau unter welchen Umständen zu tun ist. Am Ende geht es um den Einzelfall, um das, was die spezifische Open-Source-Lizenz konkret einfordert. Trotzdem ist es gelegentlich hilfreich, aus einer Übersicht – auch wenn sie nicht beansprucht, alle Details darzustellen – die Richtung zu entnehmen, aus der man Anforderungen erwarten darf:

Klasse	Lizenztext	Extra-Dokumente	Komponenten-Code	System-Code
Permissive Lizenzen	muss mitgeliefert werden	müssen fallweise mitgeliefert werden	kann zugänglich gemacht werden, muss aber nicht	kann zugänglich gemacht werden, muss aber nicht
Lizenzen mit schwachem Copyleft-Effekt	muss mitgeliefert werden	müssen fallweise mitgeliefert werden	muss zugänglich gemacht werden	kann zugänglich gemacht werden, muss aber nicht
Lizenzen mit starkem Copyleft-Effekt	muss mitgeliefert werden	müssen fallweise mitgeliefert werden	muss zugänglich gemacht werden	muss zugänglich gemacht werden

80 Die EUPL ist in mehrfacher Hinsicht ein Sonderfall. So wird sie als mehrsprachig veröffentlicht, u. a. in Deutsch https://eupl.eu/1.2/de/. Sie enthält zudem eine explizite »›*Copyleft*‹-*Klausel*« (s. § 5) und wird i. d. R. dem starken Copyleft zugeordnet (vgl. https://www.ifross.org/artikel/public-license-eupl-22-sprachfassungen-verfuegbar). Allerdings enthält sie auch eine »Kompatibilitäts-Klausel«, die besagt, dass im »Konfliktfall« die Verpflichtungen der anderen Lizenz »Vorrang« haben, sofern diese andere Lizenz in der Liste der »kompatiblen Lizenzen« erwähnt ist. Und in dieser Liste erscheint dann auch die ›EPL‹ – eine Lizenz mit schwachem Copyleft. (vgl. dazu auch: https://telekom.github.io/oslic/releases/oslic.pdf S. 33 f)

Dies kann in **Daumenregeln** überführt werden, und zwar

- **für die permissiven Lizenzen:**
  - Distribuieren Sie den Lizenztext zusammen mit dem Paket (Produkt), das die Software enthält. Der Lizenztext muss der sein, der wirklich zur Software gehört, kein Lizenztemplate. Und distribuieren Sie ihn tatsächlich zusammen mit der Software, nicht sekundär per Downloadlink.
  - Distribuieren Sie außerdem die Dateien zusammen mit dem Paket, deren Distribution die Lizenz fordert.
  - Unterlassen Sie das, was die Lizenzen Ihnen untersagen.

- **für die Lizenzen mit schwachem Copyleft-Effekt:**
  - Distribuieren Sie den Lizenztext zusammen mit dem Paket (Produkt), das die Software enthält. Der Lizenztext muss der sein, der wirklich zur Software gehört, kein Lizenztemplate. Und distribuieren Sie ihn tatsächlich zusammen mit der Software, nicht sekundär per Downloadlink.
  - Distribuieren Sie außerdem die Dateien zusammen mit dem Paket, deren Distribution die Lizenz fordert.
  - Machen Sie selbst auch den Source Code der von Ihrem System genutzten Komponente zugänglich. Ein Verweis auf bestehende Repositories genügt nicht. Und machen Sie den Code von exakt der Version zugänglich, die in Ihrem System eingebaut ist.
  - Unterlassen Sie das, was die Lizenzen Ihnen untersagen.

- **für die Lizenzen mit starkem Copyleft-Effekt:**
  - Distribuieren Sie den Lizenztext zusammen mit dem Paket (Produkt), das die Software enthält. Der Lizenztext muss der sein, der wirklich zur Software gehört, kein Lizenztemplate. Und distribuieren Sie ihn tatsächlich zusammen mit der Software, nicht sekundär per Downloadlink.
  - Distribuieren Sie außerdem die Dateien zusammen mit dem Paket, deren Distribution die Lizenz fordert.
  - Machen Sie selbst auch den Sourcecode der von Ihrem System genutzten Komponente zugänglich. Ein Verweis auf bestehende Repositories genügt nicht. Und machen Sie den Code von exakt der Version zugänglich, die in Ihrem System eingebaut ist.
  - Machen Sie schließlich auch den Sourcecode des von Ihnen entwickelten übergeordneten Systems zugänglich, das die Komponente mit starkem Copyleft nutzt. Ein Verweis auf existierende, von Ihnen bereitgestellte Repositories genügt nicht. Und machen Sie auch hier den Code von exakt der Version zugänglich, der in Ihrem System eingebaut ist.
  - Unterlassen Sie das, was die Lizenzen Ihnen untersagen.

- **für alle Lizenzen:**
  - Beachten Sie sorgfältig die lizenzspezifischen *Dos and Don'ts*
  - Gründen Sie Ihre Compliance-Analyse nicht auf Anleitungen, die den Gegenstand unangemessen verdichten – wie diese Daumenregeln es tun.

7.3 Open-Source-Compliance-Tools

Wir sehen, dass einiges getan werden muss, um ein Open-Source-basiertes Produkt lizenzkonform anzubieten. Und das wird unter komplexen Bedingungen getan werden müssen. Zuletzt läuft es ja immer darauf hinaus, die richtigen Compliance-Artefakte im richtigen Kontext zu aggregieren, zu generieren und so in das Paket/Produkt zu integrieren, dass die Kundinnen und Kunden sie einsehen können. Diese Aufgabe schreit nach Unterstützung, in thematischer und technischer Hinsicht.

7.3.1 Schulung

Eine zentrale deutschsprachige Quelle in Sachen *Open-Source-Compliance* ist das Buch »Open-Source-Software. Rechtlich Rahmenbedingungen der Freien Software«. Es erläutert juristische Zusammenhänge und historische Hintergründe.[81]

Daneben steht als Tool auch »ifrOSS«, das »Institut für Rechtsfragen der Freien und Open-Source-Software«[82], das sich eigenem Bekunden nach die Aufgabe stellt, »[...] die rasante Entwicklung der Freien Software in rechtswissenschaftlicher Hinsicht zu begleiten«[83]. Ist der Vorteil des Buches seine systematische Verdichtung, dann besteht der Gewinn dieser im Internet zugänglichen Site in seiner Vielfalt und Aktualität.

Als ein drittes Tool bietet das OpenChain-Projekt umfangreiches übergreifendes Schulungsmaterial an.[84] Zu Open-Source-Lizenzen findet sich Material auf den Seiten der *OSI*, der *Open-Source-Initiative*[85] und zu spezifischen Lizenzen empfiehlt es sich, auf die Seiten der Organisationen zurückzugreifen, die die jeweilige Lizenz pflegen, sofern eine solche Organisation vorhanden ist.

81 vgl. Jaeger, Till u. Axel Metzger: Open-Source-Software. Rechtliche Rahmenbedingungen der Freien Software; 3. Aufl.; München: C. H. Beck 2011. Dieses Buch ist aktuell in der 5. Auflage erschienen und wird – so ist zu hoffen – weiter aktualisiert werden. Andere inspirierende Quellen könnten zum Beispiel das Handbuch »Open Source for Business« sein [vgl. Heather/Meeker, »Open Source for Business – A Practical Guide«, 3. Auflage 2020], die Kommentierungen der wichtigen Open-Source-Lizenz GPL im Urheberrechtskommentar [vgl. Fromm/Nordemann/Czychowski, Urheberrecht, 12. Auflage 2018, GPL, ab Seite 2779] oder Bücher zum Urheberrecht allgemein [vgl. etwa: Wandtke/Bullinger/Grützmacher, UrhR, 5. Auflage 2019, § 69c Rn. 73 ff.

82 vgl. https://www.ifross.org/

83 vgl. ifrOSS: Ziele, Aufgaben, Geschichte https://www.ifross.org/?q=node/16

84 siehe https://www.openchainproject.org/resources

85 vgl. https://opensource.org/

7.3.2 Beratung

Mittlerweile bieten sich den Nutzerinnen und Nutzern, die Open-Source-Software in ihr Produkt einbinden wollen, eine Reihe von Personen und Institutionen an, die ihnen bei der Herstellung der Open-Source-Compliance unter die Arme greifen. Das beginnt nicht erst bei namhaften, auch international anerkannten Juristinnen und Juristen, reicht über etablierte Firmen, die einen Full-Service anbieten und führt schließlich zum Open-Chain-Projekt[86], das sich dem Thema der weltweiten Lieferketten-übergreifenden Open-Source-Compliance verschrieben hat. Dies näher zu beschreiben, hat seiner übergeordneten Rolle wegen einen besonderen Wert:

Als Projekt ist das bereits genannte *OpenChain* der *Linux Foundation* angegliedert.[87] Es bildet im Kern »a diverse, global commercial partner network«, über das sichergestellt werden soll, dass jede Organisation jeder Größe den Industriestandard anwenden kann, den OpenChain entwickelt.[88] Bestand diese Norm zunächst in der Konzeption einer idealen Struktur, gegen die sich Firmen selbst zertifizieren konnten,[89] so ist es *OpenChain* zuletzt gelungen, ihren Standard offiziell zum »International Standard for open-Source-license compliance« mit der Nummer *ISO 5230* erheben zu lassen.[90]

Die Grundidee von *OpenChain* ist mithin, dass die Verwendung von Open-Source basierten Zulieferprodukten um so einfacher wird, je mehr Zuliefernde sich an diesen Standard halten. Dass der Weg zu einer unkontrollierten Übernahme von Vorprodukten samt passender Compliance-Artefakte trotzdem noch weit ist, versteht sich von selbst.

Neben seinen zentralen Aufgaben verwaltet *OpenChain* eine Reihe von ›work groups‹, darunter auch das ›Subprojekt‹ *Open-Source-Tooling for Open-Source-Compliance*, das von sich sagt, es sei »[...] focused on reducing reSource costs and improving the quality of results around open Source compliance activities.«[91] Tatsächlich steht hier die Verknüpfung existierender Open-Source-Tools zu einer geschlossenen ›Tool-Chain‹ im Zentrum, mittels derer die Compliance-Artefakte, die man ihrem Open-Source basierten Produkt beilegen muss, automatisch generiert werden. Zu diesem Zweck pflegt diese Gruppe auch eine ausgezeichnete Übersicht über Programme, die die Generierung von Compliance-Artefakten unterstützen.[92]

86 vgl. https://www.openchainproject.org/
87 vgl. https://www.linuxfoundation.org/projects
88 vgl. https://www.openchainproject.org/partners
89 vgl. https://www.openchainproject.org/get-started/conformance
90 vgl. https://www.openchainproject.org/
91 vgl. https://oss-compliance-tooling.org/
92 vgl. https://oss-compliance-tooling.org/Tooling-Landscape/OSS-Based-License-Compliance-Tools/

7.3.3 Tools

Ruft man sich in Erinnerung, was getan werden muss, um sein Produkt lizenzkonform aufzubereiten, sieht man leicht, dass die anstehenden Arbeiten typisiert werden können:[93]

- Der **Analyzer** stellt die Abhängigkeiten fest: Man muss wissen, welche Open-Source-Software insgesamt zu einem Produkt dazugehört.
- Der **Downloader** sichert den Inhalt der entsprechenden Repositories so auf dem lokalen System, dass dieser hinsichtlich der Lizenzen gesichtet werden kann.
- Der **Scanner** stellt dann die tatsächlichen Lizenzierungen fest und markiert Elemente, die in Compliance-Artefakte eingebaut beziehungsweise umgewandelt werden können.
- Der **Evaluator** erlaubt es, eigene Nutzungsregeln in den Prozess einzubringen und anwenden zu lassen.
- Die **Reporter** aggregieren die Informationen zu den je benötigten Compliance-Artefakten.

Je nach intendiertem Zweck können verschiedene Reporter definiert und eingebunden werden.

> Man kann die Herstellung der Compliance mithin noch nicht komplett automatisieren.

Wenn man verstehen möchte, welche Tools einsatzbereit sind und in dieser Aufgabensuite verwendet werden können, möge man einen Blick auf die *OpenChain-Tooling-Landscape* werfen.[94] Und wenn man sehen möchte, inwieweit diese bereits tatsächlich zu Ketten verknüpft werden können, möge sie *ORT*[95] oder *QMSTR*[96] zu Rate ziehen. Allerdings gibt es derzeit noch keine durchgehende Kette. Man kann die Herstellung der Compliance mithin noch nicht komplett automatisieren. Die gute Nachricht ist, dass die Open-Source-Community auch daran schon arbeitet.

93 Wir folgen hier – leicht modifiziert – der Terminologie *vom ORT*, dem *Open-Source-Reviewtoolkit*. Es ist ein ›Metasystem‹, das für die einzelnen Compliance-Aufgaben so weit als möglich bereits existierende Open-Source-Lösungen heranzieht. (vgl. https://github.com/oss-review-toolkit/ort). Als übergeordnetes Tool zur Erzeugung von Compliance-Artefakten kommt ORT schon sehr weit, zumal es sehr flexibel konfiguriert werden kann. Allerdings arbeitet es noch mit einem One-Fits-All-Ansatz. Was fehlt, ist eine Komponente, die juristisches Lizenzwissen mitbringt und auf den je beobachteten Einzelfall anwendet, sodass je nach Kontext passende Artefakte entstehen. An diesem Desiderat wird mit OSCake, der *Open-Source-Compliance artifact knowledge engine* gearbeitet. (vgl. https://github.com/Open-Source-Compliance/OSCake). Wenn das umgesetzt worden ist, wird die Compliance wirklich automatisiert sein.

94 vgl. https://oss-compliance-tooling.org/Tooling-Landscape/OSS-Based-License-Compliance-Tools/

95 vgl. https://github.com/oss-review-toolkit/ort

96 vgl. https://qmstr.org/

7.4 Besondere Herausforderungen

Viele Open-Source-Lizenzen stammen aus dem letzten Jahrhundert oder aus den frühen 2000er-Jahren[97]. Damals gab es noch kein ›digitales Biotop‹. Computer wurden anders genutzt als heute: Wer ein Programm verwenden wollte, besorgte sich eine Kopie passend zum eigenen Betriebssystem. Dazu wurden ganze Distributionen auf Disketten vertrieben oder CDs mit Programmsammlungen Computerzeitschriften beigelegt.

So triggert noch heute – trotz der neuen Nutzungsweisen – die Weitergabe der Software die Einhaltung von Open-Source-Lizenzverpflichtungen. Die gängigen Lizenzen zielen sprachlich immer noch auf das ab, was zu ihrer Entstehung aktuell war. Das bewirkt gelegentlich eine Diskrepanz zwischen moderner Technik und traditionellen Lizenzen. Dennoch sind wir gehalten, auch diese ›alten‹ Lizenzen einzuhalten. Sie zu ignorieren, ist keine Option.

Aus dem Dilemma hilft nur das ›Mitdenken‹ der im folgenden dargestellten besonderen Herausforderungen.

> Die Tatsache der Softwareweitergabe ist auch heute noch der wesentliche Trigger für die Aktivierung von einzuhaltenden Lizenzforderungen.

7.4.1 SPDX und die Lizenzbenennung

Die Fülle der verschiedenen Lizenzen und ihrer Varianten beschwert gelegentlich den Dialog darüber, um welche Lizenz es geht. Diesem Problem hat sich *SPDX* gewidmet und die Namen standardisiert[98]. Dieser Ansatz hat sich so etabliert, dass die *SPDX-Identifier* mittlerweile auch im Sourcecode verwendet und damit – so weit nicht nachträglich hinzugefügt – zum Teil des *Licensing Statements* werden.

7.4.2 Die Javascript-Herausforderung

Javascript gilt als »leichtgewichtige, interpretierte oder JIT-übersetzte Sprache«.[99] Solche Programme werden oft – aber nicht nur – als Teil eines Webfrontends an Computer gesendet, auf dem sie ein in die Browser integrierter Javascript-Interpreter

97 Die BSD-Lizenzen sind in den frühen 1980er Jahren zum ersten Mal angewendet worden (vgl. http://www.lin-fo.org/bsdlicense.html); die MIT-Lizenz stammt von 1988 (vgl. https://de.wikipedia.org/wiki/MIT-Lizenz), hat aber auch eine bewegte Vorgeschichte (vgl. https://opensource.com/article/19/4/history-mit-license). GPL-2.0 und LGPL-2.0 datieren ins Jahr 1991 (vgl. https://opensource.org/licenses/GPL-2.0 u. https://opensource.org/ licenses/LGPL-2.0), LGPL-2.1 ins Jahr 1999 (vgl. https://opensource.org/licenses/LGPL-2.1), Apache-1.1 ins Jahr 2000 (vgl. https://www.apache.org/licenses/LICENSE-1.1), Apache-2.0 ins Jahr 2004 (vgl. https://www.apache. org/licenses/LICENSE-2.0) und die (L)GPL-3.0-Lizenzen aus dem Jahr 2007 (vgl. https://opensource.org/licenses/GPL-3.0 u. https://opensource.org/licenses/LGPL-3.0)

98 vgl. https://spdx.org/licenses

99 vgl. https://developer.mozilla.org/de/docs/Web/JavaScript

ausführt. Viele Javascript-Bibliotheken sind MIT lizenziert – einfach, weil der Sourcecode implizit sowieso immer mitgeliefert wird. Dieser Lizenz gemäß muss der Lizenztext in diese Bibliothek ›inkludiert‹ und mit ihr distribuiert werden.[100]

Um Bandbreite zu sparen, hat sich jedoch der Standard der ›minification‹, herausgebildet, der aus dem Sourcecode alle Whitespaces und alle Kommentare entfernt.[101] Damit wird die Diskrepanz erkennbar: Die Erfüllung der Lizenz fordert etwas, was die Technik vernünftigerweise nicht umsetzt.

Javascript ist – technisch gesehen – immer quelloffen. Aber der Code ist nicht notwendigerweise Open-Source-Software. Darüber entscheidet die Lizenzierung.

Wie können auf Compliance fokussierte Personen damit umgehen? Sie könnten zum Beispiel die nicht-minimierten Version verwenden. Das würde auf Dauer das Netz ausbremsen. Oder sie könnten in ihren Versionen den Lizenztext vor der Auslieferung wieder einführen. Das würde eine aufwendige Integration in die eigene CI/CD-Pipeline bedingen. Oder sie könnten die Lizenztexte selbst zum Download bereitstellen. Damit würden Code und Lizenz eben doch nicht als Einheit versendet. Damit entsteht eine im klassischen Sinne tragische Situation: Was auch immer man tut, es ist falsch.

Glücklicherweise entspannt sich die Lage insofern, als dass die Entwicklerinnen und Entwickler der Bibliotheken selbst diese als minimierte Versionen anbieten und damit das Verfahren implizit zu billigen scheinen. Zudem unterstützen die meisten ›Minifier‹ inzwischen das Erhalten von für die Lizenz relevanten Kommentaren.

7.4.3 Die AGPL oder die Netznutzung als Compliance-Trigger

Die AGPL soll freie Software auch in Zeiten der Cloud-Technologie gewährleisten.

Wie erwähnt, haben die Open-Source-Lizenzen ein gewisses Alter und knüpfen technisch an den Stand an, der zu ihrer Entstehung aktuell war. Deshalb triggert üblicherweise die Weitergabe der Software die Einhaltung der Compliance-Regeln. In Zeiten von webbasierten Services, wo nicht mehr Programme, sondern Daten ausgetauscht werden, werden mithin auch weniger Compliance-Fälle angestoßen: Die Open-Source-Software wird als Service in der Cloud oder im Rechenzentrum betrieben und wird eben nicht weitergegeben. Das ist formal richtig, entspricht aber nicht ganz der Idee freier Software.

Um diese Lücke zu schließen, ist die AGPL-3.0 entwickelt worden. Sie gleicht mit einer einzigen Ausnahme der GPL-3.0. Der hinzugefügte § 13 legt fest, dass auch den Nutzerinnen und Nutzern gegenüber alle Pflichten der Lizenz erfüllt werden müssen, die mit dem Programm »remotely through a computer network« interagieren.[102]

100 In der MIT-Lizenz heißt es explizit »The above copyright notice and this permission notice **shall be included in all copies or substantial portions of the Software**.« vgl. https://opensource.org/licenses/MIT [herv. KR.]

101 vgl. https://www.cloudflare.com/de-de/learning/performance/why-minify-javascript-code/

102 vgl. https://opensource.org/licenses/AGPL-3.0, § 13. Allerdings kompliziert sich die Lage noch etwas: Zum einen begrenzt die AGPL den Trigger hier auf die Fälle, wo der Serviceanbieter das Programm verändert hat. Zusätzlich fordert sie dann aber von ihm – in einer sprachlich eher ungenauen Weise –, dass auch alle zusätzlichen Teile unter derselben Lizenz distribuiert werden müssen. Unwillige Interpreten könnten darunter den ganzen Softwarestack verstehen.

Wer also Services im Netz offeriert, sollte trotzdem wachsam sein. Es reicht nicht, sich seine Lizenzkonformität aus der Tatsache herzuleiten, dass die Software nicht weitergegeben wird. Vielmehr gilt es, alle AGPL-lizenzierten Komponenten gesondert zu analysieren.

7.4.4 (L)GPL-v3 und die Ersetzbarkeit

Auch die 3. Generation der (L)GPL-Lizenzen enthält die zentrale Komponente aller GNU-Lizenzen, den Copyleft-Effekt. Zur Zeit der Überarbeitung der 2. Generation[103] – 2005 bis 2007 – gab es zwei andere, bedrängende Themen in der Open-Source-Community, nämlich den Umgang mit Software-Patenten[104] und dem ›Digital-Rights-Management‹.

In Sachen DRM fordert die GPL-3.0 zum einen,[105] dass jemand, der so lizenzierte Software in Systemen einsetzt, »[...] auf sein Recht verzichtet, die Umgehung technischer Schutzmaßnahmen zu verbieten«.[106] Zum anderen verlangt die GPL-3.0,[107] »[...] dass beim Vertrieb von Geräten mit GPL-Software die erforderlichen Informationen zur Installation [...] mitgeliefert werden müssen, um bearbeitete GPLv3-Programme wieder aufspielen zu können«.[108]

Beide Festlegungen zusammen haben dramatische Auswirkungen auf die Verwendung von (L)GPL-3.0 lizenzierter Software in Geräten: Wird sie verwendet, muss deren Hersteller es allen, die so ein Gerät erhalten, technisch und/oder prozessual ermöglichen, verbesserte Versionen kompilieren und darauf aufspielen zu können. Und er darf die Nutzung dieser technischen Möglichkeit auch nicht vertraglich einschränken. Der Hersteller darf es seinen ›Kundinnen und Kunden‹ nicht einmal verbieten, diese Geräte zu ›hacken‹, um verbesserte Versionen aufspielen zu können.

> Die (L)GPL-3.0 lizenzierte Software bedarf der besonderen Sorgfalt, wenn sie in »Embedded-Systems« integriert werden soll.

Diese Vorgaben treffen IoT-Systeme mit (L)GPL-3.0 Software ebenso wie Autos, Eisenbahnen, Waschmaschinen und viele mehr.

Die Produktverantwortlichen müssen mithin gründlich bedenken, wie mit (L)GPL-3.0 lizenzierter Software im geschlossenen System umgegangen werden soll. Sie könnten sie generell aus den Produkten verbannen, denen sie kein Interface zur Ersetzung spendieren wollen. Oder sie könnten ihr Gerät mit einer solchen Schnittstelle designen lassen. Oder sie könnten der Idee nachgehen, die Eigenentwicklungen ihrer Kundinnen und Kunden von ihrem eigenen Team einspielen zu lassen. Täten sie das, wären sie gut beraten, die Gewährleistung gesondert zu regeln und für gesetzlich abgenommene Geräte Nachzertifizierungen vorzusehen. Beim Autotuning ist das im Verbund mit dem TÜV ein gängiges Verfahren.

Wie auch immer, in ›geschlossenen‹ Systemen gilt es, die (L)GPL-3.0 lizenzierter Software besonders im Auge zu behalten.

103 vgl. Jaeger, Metzger: Open-Source-Software. München 2011, S. 50 ff.
104 vgl. https://www.fsf.org/blogs/community/the-threat-of-software-patents-persists und https://opensource. org/licenses/GPL-3.0, § 11
105 vgl. https://opensource.org/licenses/GPL-3.0, § 3
106 vgl. Jaeger, Metzger: Open-Source-Software. München 2011, S. 61.
107 vgl. https://opensource.org/licenses/GPL-3.0, § 6
108 vgl. Jaeger, Metzger: Open-Source-Software. München 2011, S. 61.

7.4.5 Open-Source-Compliance, automatische Updates und CI/CD-Chains

Heute ist es üblich, Firmwareupdates über das Netz einzuspielen. So ein neu zusammengestellter Stack muss hinsichtlich der Open-Source-Compliance natürlich erneut überprüft werden. Oft verlangt diese neu zu generierende Compliance-Artefakte. Denn Lizenzen können sich von der einen zur nächsten Version ändern, es können Copyrightlines hinzugekommen und mitzugebende Dateien angepasst worden sein.

Damit entsteht eine besondere Herausforderung für die Produktpflege: Wer eine CI/CD-Pipeline unterhält, um Updates für seine Produkte zu generieren und aufzuspielen, sollte die Generierung der Compliance-Artefakte darin integrieren. Und wer neue Versionen ›manuell‹ erzeugt, wird auch die Compliance-Aufbereitung ›manuell‹ wiederholen wollen.

Wer eine CI/CD-Pipeline unterhält, um Updates für seine Produkte zu generieren und aufzuspielen, sollte die Generierung der Compliance-Artefakte darin integrieren.

7.4.6 Maven oder die automatische Paketaggregation

Maven[109] ist der Prototyp für eine spezielle Compliance-Herausforderung: Die Grundidee dieses Tools ist es, dass die Entwicklerinnen und Entwicklerin einer POM-Datei[110] die Komponenten ihres Projektes definieren und dass *Maven* auf dieser Basis den Buildprozess eigenständig und reproduzierbar ausführt.[111] Dazu werden ›on the fly‹ die je benötigten FOSS-Komponenten aus einem (externen) Repository in die lokale Entwicklungsumgebung hineingeholt.[112]

In der POM dürfen die Komponenten auch ›unterspezifiziert‹ definiert werden: Fehlt die Releasenummer, holt *mvn* irgendeine Version. Fehlt die Location, holt *mvn* die Komponenten irgendwo her und so weiter. Das vereinfacht die Entwicklungsarbeit und beschwert die Compliance-Supervision. Denn es bleibt vorderhand systemisch unklar, welche Software in ein Produkt wirklich eingebaut worden ist. Selbst wenn die POM-Datei ausführlich formuliert worden ist, können Netzirritationen *maven* immer noch veranlassen, ›ausnahmsweise‹ ein anderes Repository zu benutzen. Damit muss der Inhalt der referenzierten Pakete – auch im Hinblick auf mitgelieferte Compliance-Artefakte – nicht mehr gleich sein. Erarbeitet man die FOSS-Compliance also auf einem bestimmten *maven*-Stand, kann der nächste Aufruf von *mvn build* die Basis der Compliance-Arbeit schon wieder verändert haben.

109 vgl. https://maven.apache.org/what-is-maven.html
110 vgl. https://maven.apache.org/guides/introduction/introduction-to-the-pom.html
111 vgl. https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
112 vgl. https://maven.apache.org/repository/index.html

Auch hier gibt es wenigstens Lösungsansätze und -ideen:

Man könnte ihr Programm beispielsweise radikal als ›Eigenentwicklung + Manifest‹ vertreiben. Die garantierte Reproduzierbarkeit sollte sicherstellen, dass der Buildprozess auch auf anderen Systemen durchläuft. Für die Komponenten, die man auf dem Kundenrechner zusammensucht, bräuchte der Entwickler seinen Kundinnen und Kunden gegenüber keine Compliance-Artefakte zu generieren. Gewissermaßen gibt nicht er die Software an die Kundinnen und Kunden, sondern *mvn*. Also wäre der Entwickler nur für die lizenzkonforme Übergabe seiner eigenen Entwicklung zuständig. Dieser Ansatz hat aber mindestens den Nachteil einer sehr erschwerten Gewährleistung: Denn wie will eine Firma für etwas geradestehen, das zuletzt auf den Kundenrechnern zusammengefügt wird? Klappt etwas nicht, werden sich die Kundinnen und Kunden doch wieder an ihren Lieferanten wenden, selbst wenn die Ursache für den ›Fail‹ auf seinem Kundenrechner liegt. Und der Produzent hat mindestens den Aufwand, dies zu ermitteln und zu belegen.

> Wann immer ein Entwickler sich seine Softwarepakete transparent zusammenstellen lässt, muss die Open-Source-Compliance-Arbeit besonders konzipiert werden.

Alternativ könnte der Entwickler dazu übergehen, sich die Komponenten in ein eigenes ›Firmen‹-Repository zu holen und die Manifeste zuletzt darauf referieren lassen. Damit hätte er den Bestand unter Kontrolle und eine solide Basis für seine Compliance-Tätigkeiten, müsste aber einen erhöhten Aufwand in Sachen Repository- und Versionspflege in Kauf nehmen.

Maven ist nicht die einzige Aggregationsautomatik. Für andere Ansätze gilt dasselbe: Wann immer ein Entwickler sich seine Softwarepakete transparent zusammenstellen lässt, muss die Open-Source-Compliance-Arbeit besonders konzipiert werden. Denn die Lizenzanforderungen zu ignorieren, ist keine Option.

7.4.7 Compliance in der Cloud: VM

Auch Clouds, die mit virtuellen Maschinen operieren, haben ihre spezielle Compliance-Herausforderung. Ihre Architektur sieht – unter dem Vorbehalt, dass es angepasst wird – so aus:

- Auf dem untersten Level stehen die eigentlichen Rechner, auf denen unter Umständen ein FOSS-Betriebssystem wie *Linux* läuft.
- In der Mitte fasst dieser Rechner ein Virtualisierungslayer – wie etwa Openstack[113] – zur eigentlichen Cloud zusammen. Und dieser Layer kann auch Open-Source-Software nutzen: Openstack wird beispielsweise unter der Apache-2.0 distribuiert.
- Zuletzt gehört es bei solchen Systemen dazu, dass der Virtualisierungslayer den angeforderten virtuellen Maschinen nicht nur Speicher und Plattenplatz zur Verfügung stellt, sondern auch je ein startbares Softwareimage. Ohne dies könnte es die virtuelle Maschine gar nicht hochfahren.

113 vgl. https://www.openstack.org/

Vom Blickwinkel der Compliance-Verantwortlichen stellt sich die Lage so dar:

- Dass die Rechner an sich den Cloudbetreibern lizenzkonform übergeben werden, ist Aufgabe der Hardwarelieferanten.
- Dass der Virtualisierungslayer den Cloudbetreibern lizenzkonform ausgehändigt wird, ist Aufgabe derjenigen, die ihn auf der Hardware installieren.
- Mit den Images wird Software an die Nutzerinnen und Nutzer einer virtuellen Maschine übergeben, zumindest dann, wenn dieses Image einen ssh-Zugang inkludiert. Denn wer einen ssh-Zugang hat, wird in der Regel auch einen scp-Zugriff haben und kann sich damit den Softwarebestand seiner virtuellen Maschine herunterladen.

Damit muss bei einer Cloud, die mit virtuellen Maschinen operiert, derjenige für die lizenzkonforme Aufbereitung der Images sorgen, der das Image bereitstellt. Das kann derjenige sein, der eine virtuelle Maschine ordert – sofern er sein eigenes Image mitbringt und in den Abrufprozess von Openstack integrieren lässt. In den meisten Fällen werden die Images aber als Bestandteil der Virtualisierungsschicht von dem Cloudbetreiber bereitgestellt. Damit ist er seinen Kundinnen und Kunden gegenüber zuständig, die Compliance-Artefakte für alle in den Images verwendeten Open-Source-Komponenten zu generieren und mitzugeben.

7.4.8 Das starke Copyleft ohne starkes Copyleft

»Exceptions« sind eine weitere Methode, den starken Copylefteffekt zu begrenzen.

Bibliotheken, die unter einer Lizenz mit starkem Copyleft verteilt werden, erzwingen, dass die Arbeiten, die darauf aufsetzen, unter derselben Lizenz weitergegeben werden. So weit, so bekannt, so unkritisch.

Merkwürdig wird es, wenn eine zentrale Kernbibliothek auf der untersten Ebene eines Softwarestacks so lizenziert wird. Auch die Entwickler des GNU Systems haben diese Auswirkung bedacht und darum die glibc, also die Library, die Aufrufe in den Kernel ermöglicht und von jeder anderen Komponente zwingend genutzt wird,[114] unter die LGPL gestellt,[115] nicht unter die GPL.

Anders bei OpenJDK[116]: Diese freie Java-Standard-Bibliothek ist im Kern unter der GPL-2.0 lizenziert. Und wie die libc wird sie von allen Java-Programmen verwendet, sofern sie als Java-Standard-Bibliothek installiert ist. Dies könnte auf einen Clash hindeuten. Tatsächlich ist OpenJDK unter der *GPL-2.0-with-classpath-exception* veröffentlicht: Die bekräftigt zwar den allgemeinen starken Coypleft-Effekt, gewährt für diese Bibliothek aber eine Ausnahme[117] und reduziert den Effekt auf ein schwaches Copyleft.

114 vgl. https://www.gnu.org/software/libc/libc.html
115 vgl. https://directory.fsf.org/wiki/Libc#tab=Details
116 vgl. https://openjdk.java.net/
117 vgl. https://openjdk.java.net/legal/gplv2+ce.html

Bei der Compliance muss man also immer auch im Hinterkopf haben, dass nicht jede Lizenz mit starkem Copyleft unter allen Umständen den starken Copyleft-Effekt auch auslöst.

7.4.9 Upstream Compliance

Veröffentlicht man etwas als Open-Source – dafür hat sich mittlerweile der Terminus ›Etwas upstream geben‹ eingebürgert –, trifft sie auf zwei Varianten:

Zum einen kann man ein eigenes Projekt mit eigenem Repository aufsetzen. In diesem Fall wird die Freiheit der Lizenzwahl nur durch den starken Copyleft-Effekt vorausgesetzter Komponenten begrenzt. Unabhängig davon soll mit der Veröffentlichung einhergehen, dass andere Nutzerinnen und Nutzer die Bedingungen unser Lizenzierung erfüllen. Also empfiehlt es sich, die von der gewählten Lizenz geforderten Artefakte selbst schon dem Repository beilegen. Das schafft Klarheit.

Zum anderen kann man an bestehenden Projekt mitarbeiten, also kontribuieren wollen. Dies wird nur erfolgreich sein, wenn man sich den Gepflogenheiten des Projektes unterwirft. Dies geht bis hin zur Lizenzierung und Zeichnung eines möglicherweise geforderten Contributor-License-Agreements. Für beide gibt es mit REUSE[118] ein neueres Verfahren, das die Ablage von Lizenzen und Lizenzierungsaussagen standardisiert.

7.4.10 Exportkontrolle

Die moderne Softwareentwicklung agiert grenzüberschreitend, die Auslieferung von Software erfolgt über die staatlichen Landesgrenzen hinweg. Bezogen auf das Open-Source-Phänomen wird es sich dabei oft um eine Redistribution (Weiterauslieferung, Umverteilung) handeln, wenn Bestandteile einer Software zuvor aus dem Internet heruntergeladen und im Zuge eines Produktvertriebes distribuiert werden.

Diese übliche Beschaffung von Software kann aber dazu führen, dass Bestandteile in das Produkt einfließen, die weder für die Nutzung in Deutschland zugelassen sind noch von dort aus an bestimmte Länder außerhalb Deutschlands ausgeliefert werden dürfen. Vor allem internationale Unternehmen sind gefordert, diesen Aspekt beim Aufsetzen ihrer Lieferkette zu berücksichtigen. Dazu finden sich im Netz eine Reihe von Hinweisen[119].

118 vgl. https://reuse.software/

119 vgl. etwa https://www.apache.org/foundation/license-faq.html oder https://www.cloudfoundry.org/ exports/. Wichtig sind hier die Erläuterung zur Exportklassifizierung gemäß der »Export Classification Number« (ECCN) und Hinweise zu Embargo Möglichkeiten.

7.4.11 Compliance und Softwarepatente

Patente dienen dem Schutz technischer Erfindung und räumen den Patentinhabenden eine zeitlich befristete Monopolstellung ein, wofür sie im Gegenzug die technische Lehre offenlegen müssen.

Für den Bereich der Software haben sich die Gesetzgebenden jedoch vorrangig für einen Schutz durch das Urheberrecht entschieden. So sind Computerprogramme als Sprachwerke urheberrechtlich explizit schutzfähig.[120] Hingegen werden im Patentrecht Programme für Datenverarbeitungsanlagen zunächst ausdrücklich nicht als Erfindungen angesehen.[121] Dies stellt jedoch nur klar, dass eine Umsetzung als Software nicht schon aus sich heraus als Erfindung gilt. Für die Patentierbarkeit ist entscheidend, ob ein technischer Bezug, die sogenannte Technizität vorliegt. So sind bereits die Ansprüche patentfähig, die zur Lösung eines Problems auf den klassischen Gebieten der Technik die Abarbeitung bestimmter Verfahrensschritte durch einen Computer beinhalten. Darüber hinaus kann eine schutzfähige, computerimplementierte Erfindung vorliegen, wenn sie Besonderheiten aufweist, die eine Patentierbarkeit rechtfertigen. Dies ist in Einzelfällen oft umstritten. Regelmäßig wird hingegen ein Patentschutz für reine Anwendungsprogramme abgelehnt.

> Da Software urheberrechtlich geschützt ist, operieren auch Open-Source-Lizenzen primär urheberrechtlich.

Da Software urheberrechtlich geschützt ist, operieren auch Open-Source-Lizenzen primär urheberrechtlich. Wegen eines parallel möglichen patentrechtlichen Schutzes entstehen unter Umständen Seiteneffekte. So könnten die Nutzungsrechte bei einer rein urheberrechtlich fundierten Open-Source-Lizenzierung dort durch Patente begrenzt werden, wo gegen den urheberrechtlichen Lizenznehmer aufgrund der bestehenden Patente vorgegangen wird.

Patentklauseln in Open-Source-Lizenzen dienen dazu, diesen Konflikt aufzulösen. Dazu gibt es zwei Ansätze:

- Neben den urheberrechtlichen Nutzungsrechten wird auch eine Patentlizenz eingeräumt. Beispiele für Open Source-Lizenzen mit Patentlizenz sind die
  - GPL-3.0 (und Varianten)
  - AGPL-3.0
  - Apache-2.0
  - MPL-1.1/2.0
  - EPL-1.0/CPL-1.0

- Der Open-Source-Lizenz wird eine Patentabwehrklausel beigefügt, die zwar keine patentrechtlichen Nutzungsrechte einräumt, aber im Fall eines patentrechtlichen Vorgehens die eingeräumten urheberrechtlichen Nutzungsrechte aufhebt. Beispiele für Open Source-Lizenzen-Patentabwehrklauseln sind die
  - Apache-2.0
  - MPL-1.1/2.0
  - EPL-1.0/CPL-1

120 vgl. UrhG § 2 Abs. 1 und §§ 69a
121 vgl. PatG § 1 Abs 3 und 4 und entsprechend EPÜ Art. 52 Abs. 2 und 3

Mithin sind Patentklauseln mittlerweile verbreitete Regelungsinhalte in Open-Source-Lizenzen. Überdies gibt es Lizenzen wie die MIT-Lizenz, die die Rechteeinräumung sehr weit und nicht urheberrechtsspezifisch formulieren, sodass eine Einräumung von Patentrechten mitgemeint sein könnte.

Als Daumenregel für eine lizenzkonforme Nutzung dürfen die Patentinhabenden davon ausgehen, dass sie mit der Weitergabe von Open-Source-Software implizit ein Nutzungsrecht an den Patenten ihres Patentpools gewähren, die für die Nutzung der Software notwendig sind. Das allerdings auch nur im Kontext dieser Software und nicht generell.

7.5 Die international-rechtliche Basis

Bisher wurde erläutert, was getan werden muss, um Open-Source-Software lizenzkonform zu nutzen. Ein spannende Frage ist auch, warum diese Lizenzen überhaupt zu beachten sind. Auf welcher Basis werden diese Pflichten überhaupt an uns herangetragen, wenn es um internationale Sachverhalte geht? Dieser juristischen Frage geht der folgende Abschnitt nach.

Im Rahmen der regelmäßig urheberrechtlich geprägten Fragestellungen findet zunächst grundsätzlich das sogenannte Schutzlandprinzip Anwendung. Dies besagt im Ausgangspunkt, dass sich die rechtliche Ausgestaltung des Urheberrechts nach dem Recht des jeweiligen Staates richtet, für dessen Gebiet Schutz beansprucht wird. Im Open-Source-Kontext gilt dies für zentrale Rechtsfragen wie beispielsweise, wer eigentlich als Urheber und erster Rechteinhaber an Programmierungen gilt, wie er Rechte an ihren Programmierungen Dritten einräumen kann, ob seine Rechte überhaupt übertragbar sind und wie er diese Rechte zuschneiden kann, also welchen Inhalt die eingeräumten Rechte haben.

Für alle vertragsrechtlichen Fragen findet hingegen das sogenannte Vertragsstatut Anwendung. Dieses richtet sich in Europa nach der Rom-I-Verordnung (Rom-I-VO). Dabei ist zwar gemäß Art. 3 Rom-I-VO grundsätzlich zunächst eine etwaige Rechtswahl vorrangig. Da aber viele Open-Source-Lizenzen mit großer praktischer Bedeutung (zum Beispiel GPL-, MIT-, BSD- und Apache-Lizenzen) keine Rechtswahlklausel enthalten, kommt es regelmäßig auf das Recht am Sitz des Urhebers beziehungsweise Rechteinhabers an. Denn wenn die Parteien kein Recht explizit wählen, knüpft Art. 4 Rom I-VO an den Ort an, an dem die charakteristische Leistung erbracht wird.

Dieser liegt nach herrschender Auffassung bei der Einräumung von einfachen Nutzungsrechten – wie in allen Open-Source-Lizenzen – beim Lizenzgeber, also dem Urheber. In den Anwendungsbereich des Vertragsstatuts fallen gemäß Art. 12 Rom-I-VO Fragen des Zustandekommens, der Wirksamkeit sowie Haftungs- und Gewährleistungsfragen, aber auch Auslegungsfragen. Letzteres ist beispielsweise relevant für die Auslegung der Reichweite eines Copyleft-Effektes einer Lizenz oder der Frage, ob die Lizenz auch Rechte zur Softwarenutzung im Wege eines SaaS-Angebots einräumt.

8 Ausblick

Darüber zu schreiben, dass Open-Source-Software seinen anerkannten Platz bei den Anbietern von Produkten, Software und Dienstleistungen der Informationswirtschaft, der Telekommunikation, der neuen Medien und der Digitalwirtschaft erobert hat und aus den aktuellen und zukünftigen Produkten und Innovationen nicht mehr wegzudenken ist, wirkt, gemessen am überragenden Erfolg, etwas unterbewertet. Das Konzept Open-Source-Software ist mittlerweile 40 Jahre alt und immer wieder Herausforderungen ausgesetzt gewesen. Trotzdem ist das Ökosystem immer stärker und mächtiger geworden. Waren es in den ersten Jahren hauptsächlich Enthusiasten und Freiwillige, die im privaten Umfeld Software erstellt haben, so ist das Thema heute auch im professionellen Umfeld allgegenwärtig. Es wird in Firmen mit zunehmender Selbstverständlichkeit quer durch Funktionen wie Softwareentwicklung, aber auch andere Bereiche wie Einkauf, Marketing und Personalwesen, behandelt. Die Mehrheit der Entwickler, die zu Open-Source-Software beitragen, tut dies als Teil ihrer bezahlten Arbeit[122].

> Die Mehrheit der Entwickler, die zu Open-Source-Software beitragen, tut dies als Teil ihrer bezahlten Arbeit.

Open-Source-Software ist »Mainstream« geworden und in jeder Firma und allen Geschäftsfeldern angekommen.[123]

Aber natürlich gibt es in Zukunft auch noch eine Reihe von Herausforderungen, die zu bewältigen sind, beispielsweise durch das stark wachsende Modell, Open-Source-Software als Service zu betreiben. Wenn man die Software aber nicht mehr selbst betreibt, sondern als Service konsumiert, dann wirken die Freiheiten, die durch Open-Source-Lizenzen garantiert werden, unter Umständen nur eingeschränkt für die Nutzerinnen und Nutzer der Services. Das kann dazu führen, dass sich Herstellerabhängigkeiten sogar verstärken. Gleichwohl bietet sich die Möglichkeit, Daten und Use Cases zu einem anderen Serviceprovider zu migrieren oder die Software selber zu betreiben. Entscheidend für Nutzerinnen und Nutzer ist es, die nötige Kompetenz aufzubauen, um souverän agieren zu können.

Des Weiteren besteht für Firmen, die Open Source-lizenzierte Produkte anbieten, die Gefahr, dass ihr Geschäftsmodell nicht mehr tragfähig ist, sofern es die mögliche Nutzung im Cloud-Kontext nicht berücksichtigt hatte. So haben einige Firmen auf Grund der Praktiken der sogenannten Hyper Scaler ihr Lizenzmodell geändert und setzen nicht mehr auf allgemein akzeptierte Open-Source-Lizenzen, sondern auf eigene, nicht anerkannte oder gar auf proprietäre Modelle und führen als Begründung Druck durch die Konkurrenz der großen Cloud-Provider an. Eine mögliche und bereits praktizierte Antwort des Ökosystems auf Lizenzänderungen sind in solchen Fällen ↗ Forks des jeweiligen Open-Source-Produktes.

Dieses Problem ist ein Hinweis darauf, dass die Notwendigkeit besteht, Open-Source-Lizenzen weiterzuentwickeln. Ein anderer Hinweis ist das Aufkommen sogenannter »Ethical Source Licenses«, die sich als Weiterentwicklung von Open-Source-Lizenzen verstehen, aber bestimmte, als ethisch nicht vertretbar empfundene Anwendungsfälle ausschließen. Die Herausforderung wird sein, hier die Grenzen zwischen dem, was in

122 Siehe die »FOSS 2020 Contributor Survey« der Linux Foundation. https://www.linuxfoundation.org/en/resources/publications/2020-foss-contributor-survey/

123 Open-Source-Monitor 2021, https://www.bitkom.org/openssourcemonitor

einer Software-Lizenz geregelt werden kann, und dem, was durch andere Mechanismen geregelt werden muss, klar zu ziehen und den Geist der Open-Source-Definition aufrechtzuerhalten.[124, 125]

Generell kann festgestellt werden, dass es in Zukunft sehr wichtig sein wird, zu unterscheiden, wo Open Source lediglich als schöner Marketingbegriff oder als ideologische Keule verwendet wird und wo Open Source tatsächlich das Versprechen von mehr Innovation, Effizienz, Transparenz und Unabhängigkeit einlöst.

Weitere Herausforderungen werden folgen und auch diese wird das Open-Source-Ökosystem meistern. Softwareentwicklung in und mit dem Ökosystem ist heute für viele Firmen schlicht und ergreifend notwendig, um weiterhin konkurrenzfähige softwarebasierte Produkte und Lösungen zeitgerecht auf den Markt bringen zu können. Die aktive Beteiligung im Ökosystem kann gegebenenfalls den kontinuierlich wachsenden Mangel an Entwicklerinnen und Entwicklern mildern. Gerade für Firmen aus Staaten, die mit dem demographischen Wandel konfrontiert sind, kann der »Open-Source-Weg« eine Option sein, um die Auswirkungen des Fachkräftemangels zu lindern.

Softwareentwicklung in und mit dem Ökosystem ist heute für viele Firmen schlicht und ergreifend notwendig, um weiterhin konkurrenzfähige softwarebasierte Produkte und Lösungen zeitgerecht auf den Markt bringen zu können.

Open-Source-Software hat innovative Betriebsmodelle positiv beeinflusst. Es hat sowohl bei den Beschaffungsprozessen im Unternehmen, als auch in klassischen Ausschreibungssituationen für eine kreative, verlässliche und vor allem konkurrenzfähige Nutzungserweiterung gesorgt. In den kommenden Jahren wird sich die weltweite Open-Source-Bewegung weiter der Aufgabe stellen müssen, ihren Vorsprung als ernstzunehmende Alternative und Konkurrenz zu proprietärer Software im Markt zu behaupten und vielfach weiter auszubauen. Dafür ist eine gezielte Anpassung und Modernisierung ihrer Lizenzlandschaft notwendig sowie das Erreichen höherer Standardisierungsgrade und die Etablierung eines nachvollziehbaren Qualitätsbewusstseins. Ein entscheidender Faktor ist, die teils noch aufwändige Überprüfung der Compliance mit allen ihren Aspekten durch eine strukturierte Automatisierung zu einer Selbstverständlichkeit im Software Asset Management werden zu lassen. Standards wie OpenChain ↗ ISO/IEC 5230:2020 und ↗ SPDX werden in den nächsten Jahren License Compliance entlang der Software Supply Chain vereinfachen. Doch License Compliance wird weiterhin ein spannendes Thema bleiben. Firmen sind gut beraten, entsprechende Kompetenzen aufzubauen. Als wirksames Instrument, Open-Source-Compliance in einen ganzheitlichen Open Source-Ansatz zu integrieren, hat sich das Open-Source-Program Office (OSPO) etabliert. In ihm werden alle Open-Source-Themen gebündelt: Von Compliance bezüglich der Nutzung sowie eigenen Kontributionen, bis hin zum Support von Open-Source-Themen im Business Development und dem Aufbau von Ökosystemen zur Etablierung innovativer Service-Architekturen mit Partnern und Kunden.

Bereits in der Vergangenheit wurden zahlreiche Referenzimplementierungen bzw. Defakto-Standards als Open-Source-Software realiert. Auch in Zukunft werden disruptive Lösungen auf der Basis von Open-Source-Software sich durchsetzen und einen Defakto-Standard begründen, wenn sie den Nerv der Anwenderinnen und

124 http://wiseearthtechnology.com/blo^g/peaceful-open-Source-license, https://github.com/atErik/PeaceOSL
125 https://www.golem.de/news/open-Source-lizenzen-gegen-den-missbrauch-freier-software-1509-116210.html

Anwender treffen. Allerdings bleibt zu erwarten, dass immer öfter gezielt eine Referenzimplementierung eines definierten Standards kooperativ auf der Basis von Open Source entwickelt wird. Damit bleibt zu vermuten, dass Standards in Zukunft einfacher genutzt werden können und sich damit schneller durchsetzen.

Angriffe auf die Software-Supply-Chain, bei der Schadcode direkt oder indirekt in zugelieferte Software eingebracht wird, haben in der jüngeren Vergangenheit zugenommen und dieser Trend wird sich noch massiv verstärken. Dies betrifft alle Software (und Hardware), gleich unter welcher Lizenz sie steht. Open-Source-Software bietet hier jedoch spezielle Herausforderungen, aber auch Chancen. So kann die Vielzahl von verwendeten Open-Source-Modulen in manchen Sprach-Ökosystemen es aufwändig machen, Integrität und Vertrauenswürdigkeit von verwendeten Quellen und Paketen zu überprüfen. Durch die Transparenz von Open Source lassen sich andererseits effektive Mechanismen entwickeln, um selbst Prüfungen durchführen zu können, wo bei proprietärer Software nur das Vertrauen in den Hersteller bleibt. Es ist angeraten, Sorgfalt bei der Integration von zugelieferter Software walten zu lassen sowie Community-weite Initiativen zu unterstützen, die Open-Source-Supply-Chain abzusichern. Durch die zunehmende Verbreitung von Open-Source-Software und dem steigenden Abhängigkeitsgrad werden die Anzahl und Intensität von Codereviews steigen und somit auch mehr Sicherheitslücken gefunden und behoben werden. Gleichwohl stellt der professionelle Umgang mit der »gefühlten Unsicherheit« eine Herausforderung dar.

Open-Source-Software ist in vielerlei Hinsicht ein starker Innovationsmotor. So können innovative Open-Source-Lösungen ohne nennenswertes Vorab-Investment in Bereichen eingesetzt werden, in denen sie vorher noch nicht präsent waren. Die Verfügbarkeit des Source Codes, das Recht ihn zu ändern, ihn somit an die Erfordernisse der jeweiligen Bereiche anpassen zu können und schließlich die Lizenzgebührenfreiheit von Open Source machen dies möglich. So können völlig neue Produkte und sogar Märkte entstehen. Kurz gesagt: Open-Source-Software ist ein Enabler für neue Technologien, sie ermöglicht die Erschließung neuer Märkte und die Entwicklung zukünftiger Geschäftsmodelle.

> Open-Source-Software ist ein Enabler für neue Technologien.

Während die Nutzung von Open-Source-Software inzwischen allgegenwärtig und schon zur Selbstverständlichkeit geworden ist, haben die meisten Unternehmen den nächsten Schritt noch vor sich: Die aktive Beteiligung am Open-Source-Ökosystem. Hier die nächsten Schritte zu gehen, eine Kultur des Teilens zu etablieren und pragmatisch und schließlich auch strategisch zu Open-Source-Projekten beizutragen, öffnet ein enormes Potenzial – nicht nur aus ideeller Sicht, sondern vor allem auch aus wirtschaftlicher. Die Innovationskraft und Entwicklungskapazität des Open-Source-Ökosystems kann von keiner einzelnen Firma überboten werden. Von daher ist es schlicht ein Muss, in dem Ökosystem aktiv mitzuarbeiten, um nicht technologisch auf ein Abstellgleis zu geraten. Open-Source-Foundations (vgl. ↗ Kapitel 6.4) werden hier weiterhin eine zentrale Rolle spielen, da sie durch den von ihnen bereitgestellten rechtlichen Rahmen eine neutrale Umgebung bereitstellen, in der Firmen gemeinsam Software entwickeln können. Damit sind die Voraussetzungen gegeben, dass

konkurrierende Firmen in nicht wettbewerbsrelevanten Bereichen auf der Basis von Open-Source-Software kooperieren.

> Dicebat Bernardus Carnotensis nos esse quasi nanos gigantum umeris insidentes, ut possimus plura eis et remotiora videre, non utique proprii visus acumine, aut eminentia corporis, sed quia in altum subvehimur et extollimur magnitudine gigantea.

»Bernhard von Chartres sagte, wir seien gleichsam Zwerge, die auf den Schultern von Riesen sitzen, um mehr und Entfernteres als diese sehen zu können – freilich nicht dank eigener scharfer Sehkraft oder Körpergröße, sondern weil die Größe der Riesen uns emporhebt.«

– Johannes von Salisbury um 1120: Metalogicon 3,4,47–50[126]

In diesem Sinne wünschen wir allen Firmen und Entwicklern und Entwicklerinnen den Mut, den Schritt zur Offenheit zu wagen, und sich und uns allen damit die vollen Möglichkeiten von Open Source zu erschließen.

126 https://de.wikipedia.org/wiki/Zwerge_auf_den_Schultern_von_Riesen

9 Exkurs

9.1 Zur Entstehung von Open-Source-Software

Open-Source-Software hat auch als Idee eine Geschichte,[127] sie in Grundzügen zu kennen, erleichtert das Verstehen – und dieser Aufgabe widmen sich die folgenden Abschnitte.

9.1.1 Über Unix zu Linux

Der Begriff »Open-Source-Software« selbst ist eine recht junge Erfindung. Erst 1998 – sozusagen als »Marketingkniff« – erschaffen, sollte er den bis dato gebräuchlichen Terminus »freie Software« ersetzen, der sich seit den frühen 1980er Jahren etabliert hatte. Mit dem Begriff »Free Software« setzte (und setzt) man auch politische Signale. So entstand in der Community der Wunsch nach einer pragmatischen, unverfänglichen Alternative.

Abseits dieser Programmatik war freie Software jedoch **seit den Anfängen** der kommerziellen Computertechnik präsent: In den ersten Jahrzehnten bildete freie Software bei Computerfirmen eine konstitutive Komponente ihres Geschäftsmodells, ohne allerdings schon als freie Software bezeichnet zu werden. Das eigentliche Geschäft der kommerziellen Informatik bestand anfänglich nämlich »nur« im Verkauf von Hardware. Software war das »Give-Away«, das inklusive Quellcode zusammen mit der Hardware ausgeliefert wurde. Der Austausch von verbesserten Versionen der Software unter den Kunden wurde als Verbesserung des Hardwareproduktes gesehen. Begünstigt wurde dieser Umgang durch die Art der Kundschaft: Die frühen Computersysteme waren vor allem an Universitäten beheimatet. Wissenschaftler und Techniker nutzten die mitgelieferte Software, passten sie – als Teil ihrer Arbeit – an den jeweiligen Bedarf an, gaben sie weiter, empfingen veränderte Versionen und setzten eigene Arbeiten darauf auf.

Der Austausch von verbesserten Versionen der Software unter den Kunden wurde als Verbesserung des Hardwareproduktes gesehen.

Sie begegneten damit dem Mangel an Einsatzmöglichkeiten der teuren Hardware.

Umgekehrt war es im Sinne der Firmen, dass die Software für ihre Hardware wuchs. Je größer deren Funktionsumfang war, desto stärker der Anreiz für Institute und Organisationen, sie zu kaufen.

127 Ein Buch, das Interessierte lesen sollten wenn sie sich schnell und gründlich kundig machen wollen, ist »Die Software-Rebellen« von Glyn Moody. Ein zweites ist Sam Williams Biografie »Free as in Freedom. Richard Stallman's Crusade for Free Software«. Die nachfolgende Darstellung folgt – ohne einzelne Nachweise aber mit ausdrücklichem Dank – diesen beiden Werken.

Ende der 1960er Jahre wendete sich das Blatt: IBM wurde durch eine Klage des US Department of Justice gezwungen, Hard- und Software zu entkoppeln. Software bekam einen eigenen Wert. Diese Entkoppelung legte den Grundstein für einen eigenen Markt. Betriebssysteme und Programme wurden zum relevanten Unterscheidungsmerkmal der neuen Geschäftssparte. Hardware lieferte »nur noch« die reine Rechenleistung. Software wurde zur handelbaren Ware.

In dieser Zeit entstand in den Bell Laboratories von AT&T auch die erste UNIX-Version, vornehmlich entwickelt für den internen Eigenbedarf. An die Universitäten jedoch gab AT&T ihr Unix zum Selbstkostenpreis ab und zwar samt Quellcode. Dass diese Codebasis problemlos zugänglich war, erhöhte zuerst das Benutzerinteresse. Und dieses ermunterte zu Weiterentwicklungen, genauso, wie – ironischerweise – der fehlende Support: AT&T selbst nämlich bot keinen Support für das Produkt an. Also galt es, sich selbst zu helfen, vornehmlich und am besten im Austausch mit Gleichgesinnten. So schlossen sich schnell lokale Anwender zusammen, tauschten über Vorversionen des Internets Lösungen und Verbesserungen aus und halfen bei Problemen. Über diese freie unorganisierte Kooperation in der akademischen Community erreichte UNIX zügig einen hohen Reifegrad.

Einen **ersten Schritt in Richtung Institutionalisierung** machte dann die Universität von Kalifornien in Berkeley. Sie fasste die entstehenden Verbesserungen zusammen, sammelte die neuen Unixtools und -applikationen ein und veröffentlichte alles zusammen kontinuierlich in ihrer Berkeley Software Distribution (BSD). AT&T auf der anderen Seite wandte sich mit kommerziellem Interesse der Aufgabe zu, stabile Releases zu vertreiben. Als es **1984** durch den Fall des Telefonmonopols zur Aufspaltung von AT&T kam, griff aber auch die Nachfolgefirma das Konzept einer aus sich heraus werthaltigen Software auf. Nutzer der Tools aus der BSD mussten nun auch eine AT&T-Quellcodelizenz erwerben, die immer teurer wurde. Die Kommerzialisierung von Unix schritt voran, nicht ohne eine Gegenbewegung ins Leben zu rufen. So führte die Einschränkung der freien Nutzung über die Einführung von Lizenzgebühren auch dazu, dass – nach einem entsprechenden Aufruf in einer konzertierten Aktion der BSD-Entwickler – verbliebener AT&T-Code vollständig aus dem BSD-Unix-Derivat entfernt wurde. Mit der Zeit entstand so ein wirkliches freies Unix. Wenn wir heute (neben Linux) auch auf OpenBSD, FreeBSD usw. als ein freies Unix zugreifen können, dann ist das dieser Entwicklung zu verdanken.

> Die Kommerzialisierung von Unix schritt voran, nicht ohne eine Gegenbewegung ins Leben zu rufen.

9.1.2 Ein zweiter und anderer Weg führte zu Linux

Auch an der Universität von Richard M. Stallman – dem Labor für Künstliche Intelligenz des MIT mit seiner ursprünglichen »Hacker«- und »Sharing«-Kultur in Sachen Software – machte sich die Entwicklung der Kommerzialisierung und der Abschottung praktisch bemerkbar. Gern erzählt wird die Anekdote, nach der Richard Stallman einmal seinen Drucker an neue Schnittstellen anpassen wollte, die dafür notwendige Druckersoftware aber nicht im Quellcode dem Drucker beigegeben worden war. Selbst als Stallman endlich einen universitären Kollegen gefunden hatte, der diesen Code besaß, durfte ihn der Kollege aufgrund der Lizenzvereinbarung nicht an Richard Stallman weitergeben. Software hatte auf einmal einen eigenen Wert, der durch Abgrenzung geschützt werden musste. Und eben diese Ausgrenzung – so die einfache Sichtweise von Richard M. Stallman – behindere ihn in seinem Tun.

Diese Art des Schutzes durch Abkehr von der Idee einer freien Nutzung griff bald über das kommerzielle Betriebssystem Unix hinaus. Seit 1981 in den USA Patente auf Software möglich wurden, entstand auch unter den Entwicklern selbst vermehrt der Wunsch, Programme zu besitzen. Auch von dieser Seite aus schränkte sich also die Bereitschaft zum freien Austausch von Software ein.

Stallman empfand beide Entwicklungen als eine persönliche Beschränkung. Und so versuchte er ab **1984**, die ursprüngliche Dynamik einer Kultur des Teilens, wie er sie in der frühen UNIX- und BSD-Gemeinschaft im MIT kennen und schätzen gelernt hatte, wieder zu beleben. Er begann, das **Betriebssystem GNU** (ein rekursives Akronym: »GNU«s not »UNIX«) unter den Bedingungen freier Software zu entwickeln und zu vertreiben. Mehr noch: er konzipierte die Idee freier Software und wurde so zum Vorreiter des GNU-Projekts samt der daraus entstandenen, 1985 gegründeten Free Software Foundation (FSF). GNU sollte ein komplett freies und offenes Betriebssystem sein.

Frei sollte es nicht im Sinne von »Freibier« sein, sondern im Sinne von »Freiheit«. Es sollte eine Reihe von Anwendungen umfassen, die es jedem Benutzer ermöglichen würde, die Software uneingeschränkt zu nutzen, einzusehen, zu verändern und in veränderter Form weiterzugeben.[128] Aus diesem Wunsch heraus entstand die GNU-GPL, die GNU General Public License, mit der das Konzept einer »Freien Software« erstmals in eine Lizenz gegossen wurde.

Innerhalb des GNU-Projekts entstanden danach schnell zahlreiche Anwendungen. Die garantierten Freiheiten (re-)etablierten die (universitäre) Kooperation über den Austausch. Stallmans Idee funktionierte: Die Mitarbeit des Einzelnen im Kleinen konnte sich lohnen, weil das System als Ganzes dauerhaft frei zur Verfügung stehen würde.

Dem Anspruch, ein ganz freies, vollständiges unix(artiges) Betriebssystem zu erstellen, konnte das GNU-Projekt aber nur mit einem eigenen Kernel gerecht werden. Diese

128 vgl. https://www.gnu.org/philosophy/free-sw.html

zentrale Komponente eines jeden Betriebssystems verbindet die Module und macht sie ablauffähig. Solange es zur Nutzung all der freien Tools des GNU-Projektes immer noch eines kommerziellen Unix-Kernels bedurfte, war das Ziel nicht erreicht. Linus Torvald war es schließlich, der diesen Kernel Anfang der 1990er Jahre programmierte, ihn Linux nannte und dessen Entwicklung bis heute maßgeblich beeinflusst. Er stellte seinen Linux-Kernel von Anfang an als freie Software zur Verfügung, indem er den Code unter den Bedingungen der GPL verteilte. Und auch da fruchtete die Idee von Richard Stallman: Viele weitere Informatiker auf der ganzen Welt wurden angeregt, sich zu beteiligen.

So entstand schließlich in der Kombination mit den bereits fertig gestellten Anwendungen des GNU-Projektes das, was als Ganzes heute als LINUX bekannt ist.[129]

9.1.3 Von »Free« zu »Open«

Den **Anstoß für die Entstehung der Open-Source-Bewegung** bildete **Ende der 1990er Jahre** die Offenlegung des Netscape Navigators. Netscape konnte sich damals mit dem Browser nicht gegen Microsoft und die Dominanz des Internet Explorers durchsetzen. Darum sollte die Codebasis von Netscape freigegeben und der Browser von einer freien Community weiter gepflegt werden. Heute kennen wir das Ergebnis als »Firefox« sowie das damit verbundene Mozilla-Projekt.

Doch zur Zeit dieser Freigabe eines zuerst kommerziellen Produktes stellte sich das Attribut »freie Software« in der Kommunikation mit Unternehmen als schwierig heraus, wenn man auf dieser Basis neue kommerzielle Geschäftsideen verfolgen wollte. Die mit »freier Software« verbundene Philosophie und das mittlerweile gängige Verständnis von Code als Firmengeheimnis wirkten abschreckend. Um mit den Führungsetagen und Entscheidern in Konzernen besser ins Gespräch zu kommen, wurde nach einer alternativen Bezeichnung gesucht. Geschäftsfreundlicher sollte sie klingen und weniger ideologisch behaftet daherkommen. Der Vorschlag, dafür den Namen »Open-Source-Software« zu verwenden, soll von Christine Peterson (Foresight Institute) stammen. Basierend auf diesem Namen wurde schließlich **1998 die Open-Source-Initiative (OSI)** von Bruce Perens, Eric S. Raymond und Tim O' Reilly gegründet, die Ideen der freien Software und die Debian Free Software Guidelines zur Open-Source-Definition zusammengeführt hat.[130]

Vom Gebrauch her gibt es keinen Unterschied zwischen freier Software und Open-Source-Software: Alle Lizenzen freier Software sind auch als Open-Source-Lizenzen gelistet.[131] Umgekehrt werden die vier konstitutiven Freiheiten freier Software – will sagen: sie nutzen, verstehen, weitergeben und verbessern zu dürfen[132] – vollständig

129 Das gesamte Betriebssystem wird auch als GNU/Linux bezeichnet, um zu unterstreichen, dass ein Betriebssystem nicht nur aus seinem Kernel besteht und dass die Tools aus dem GNU-Projekt einen gleich wichtigen Anteil an dem funktionierenden Ganzen haben. (Vgl. GNU-Projekt: Warum Freie Software besser ist als Open-Source-Software; https://www.gnu.org/philosophy/free-software-for-freedom.de.html

130 Open-Source-Definition http://openSource.org/osd

131 vgl. OSI: Licenses by Name http://openSource.org/licenses/alphabetical

132 vgl. GNU-Projekt: Was ist Freie Software? https://www.gnu.org/philosophy/free-sw.html

von der Open-Source-Definition abgedeckt. Die Unterschiede manifestieren sich in den Botschaften der Begriffe:

»Open-Source-Software« unterstreicht den praktischen Nutzen und die Entwicklungsmethode; »freie Software« betont die gesellschaftlichen Vorteile.[133] Pointiert ausgedrückt sei unfreie Software »[...] für die Open-Source-Bewegung [...] eine suboptimale Lösung[;] für die Freie-Software-Bewegung [... aber] ein soziales Problem und freie Software (dessen) Lösung«.[134] Jedenfalls solle man, wenn man an freie Software denke, »[...] an frei wie in Redefreiheit denken, nicht wie in Freibier.«[135]

Gelegentlich wird versucht, den Namenskonflikt über hybride Begriffe wie FLOSS (Free/Libre Open-Source-Software) und FOSS (Free and Open-Source-Software) aufzufangen.[136]

Das passt zu dem Statement, dass »die Freie-Software- und die Open-Source-Bewegung [...] so etwas wie zwei politische Lager innerhalb der Freie-Software-Gemeinschaft (seien)«.[137]

9.1.4 Generation GitHub

Die ersten Jahrzehnte von Open Source waren durch den Kampf für die neue Idee von freier und offener Software geprägt. Viele Menschen und Unternehmen nahmen das Konzept der vier Freiheiten nicht ernst, sahen es als zu idealistisch an oder bekämpften es sogar aktiv.

In der zweiten Hälfte der 2010er-Jahre änderte sich dieses Bild jedoch grundlegend. Die Idee von Open Source setzte sich durch und wurde zum Mainstream, nicht nur bei Enthusiasten und privaten Idealisten, sondern auch in der Breite der Wirtschaft. Zwei Übernahmen aus dem Jahre 2018 stellten dies überdeutlich dar. IBM übernahm den Open-Source-Hersteller RedHat für 34 Milliarden US-Dollar und Microsoft kaufte GitHub für 7,5 Milliarden US-Dollar. Dies war insbesondere deshalb bemerkenswert, da GitHub als Plattform mit damals 100 Millionen Repositories und über 30 Millionen Nutzern[138] quasi die Heimat der globalen Open-Source-Community darstellte, und es für Microsoft das Ergebnis einer radikalen Kehrtwende markierte, vom Gegner von Open Source hin zu der Firma, die weltweit im Vergleich die meisten Open-Source-Beitragenden stellt.

> Das im Scherz ausgerufene Ziel von Linux, die »World Domination« zu erreichen, ist Wirklichkeit geworden.

Die Dominanz von Open-Source-Software in allen Bereichen, in denen Software eine wesentliche Komponente darstellt, lässt sich auch an der Entwicklung von Linux

133 vgl. Stallman, Richard: Warum Open Source das Ziel Freie Software verfehlt https://www.gnu.org/philosophy/open-source-misses-the-point.html
134 GNU-Projekt: Warum Freie Software besser ist als Open-Source-Software https://www.gnu.org/philosophy/free-software-for-freedom.de.html
135 GNU-Projekt: Was ist Freie Software? https://www.gnu.org/philosophy/free-sw.html
136 vgl. Stallman, Richard: FLOSS und FOSS https://www.gnu.org/philosophy/floss-and-foss.html
137 Stallman, Richard: Warum Freie Software besser ist als Open-Source-Software https://www.gnu.org/philosophy/free-software-for-freedom.de.html
138 Siehe https://github.blog/2018-11-08-100m-repos/

ablesen. Als Hobby-Projekt[139] im Jahr 1991 gestartet, läuft es heute, im Jahre 2021, auf Milliarden von Geräten, vom Telefon[140] über den Cloud-Server[141] oder Super-Computer[142] bis zum Mars-Helikopter[143]. Das im Scherz ausgerufene Ziel von Linux, die »World Domination« zu erreichen, ist Wirklichkeit geworden.[144]

In vielerlei Hinsicht ist Open Source zur Selbstverständlichkeit geworden. Nadja Eghbal beschreibt in ihrem Forschungsbericht für die Ford Foundation ↗ »Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure« aus dem Jahre 2016, wie Open Source einen großen Teil der digitalen Infrastruktur darstellt, und dabei, wie Straßen und Brücken, als Teil unserer öffentlichen Infrastruktur gesehen werden kann. Sie beschreibt auch die Herausforderung, die bei der Pflege dieser gemeinsamen Infrastruktur entstehen kann.

In der jüngsten Generation von Software-Entwicklerinnen, für die Open Source selbstverständlich ist, die ohne viel Aufhebens Code von GitHub konsumieren, sind die Kämpfe der Vergangenheit nicht mehr präsent.

Dass es einen erheblichen Aufwand darstellt, diesen Code zu pflegen, gerät dabei manchmal in Vergessenheit. Dieses Problem zu lösen, stellt eine der Herausforderungen von Open Source für die Zukunft dar.

Man kann viel darüber diskutieren und spekulieren, wodurch dieser enorme Erfolg von Open Source verursacht wurde. Viele Aspekte werden in diesem Leitfaden beschrieben. Letztendlich kann man es wohl als eine Mischung aus natürlicher zeitlicher Entwicklung, aus harten wirtschaftlichen Vorteilen und dem Idealismus einer fantastischen Community betrachten.

139 Linus Torvalds bezeichnete Linux in seiner ersten Ankündigungs-Email in der Newsgruppe comp.os.minix als »just a hobby, won't be big and professional like gnu« (siehe https://de.wikipedia.org/wiki/Geschichte_ von_Linux#Entstehung_des_ Linux-Kernels)
140 Siehe https://www.theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021
141 Siehe https://www.zdnet.com/article/linux-now-dominates-azure/
142 Seit 2017 laufen alle 500 Super-Computer auf der TOP500-Liste unter Linux https://www.top500.org/statistics/overtime/
143 Siehe https://spectrum.ieee.org/nasa-designed-perseverance-helicopter-rover-fly-autonomously-mars 142
144 Siehe https://www.linuxjournal.com/content/25-years-later-interview-linus-torvalds

9.2 Software Bill of Materials (SBOM)

Der Begriff »Software Bill of Materials« oder abgekürzt »SBOM« ist in den letzten Jahren zunehmend präsent geworden. Er taucht im Kontext von Software-Lieferketten und Fragen zu Sicherheit, Lizenz-Compliance und ähnlichen Themen auf. Mit der Verordnung 14028 des US-amerikanischen Präsidenten aus dem Jahr 2021[145], in der die SBOM als ein Element zur Erhöhung der Cybersicherheit gefordert wird, hat das Konzept eine erhöhte Aufmerksamkeit erlangt, die sich in der gesamten Industrie zeigt.

Für Open-Source-Software ist das Konzept SBOM von besonderer Bedeutung, da Open-Source-Komponenten inzwischen einen Großteil jeder produzierten Software ausmachen. Nach dem ↗ »Synopsys Open Source Security and Risk Analysis report 2022« enthält 97 Prozent aller Software Open-Source-Komponenten und im Durchschnitt über alle Software bestehen Projekte zu 78 Prozent aus Open-Source-Software. Die große Zahl von Open-Source-Komponenten stellt hier eine besondere Herausforderung dar. Laut des Berichts ↗ »Sonatype 2021 State of the Software Supply Chain« gibt es allein in den Ökosystemen von Java, Javscript, Pyton und .NET insgesamt 37 Millionen verfügbare Open-Source-Komponenten.

In diesem Kapitel wollen wir das Konzept SBOM vorstellen, erklären, wie es angewendet werden kann, und eine Einordnung vornehmen, in welchen Bereichen und zu welchem Zweck es heute schon relevant ist und in Zukunft werden wird.

9.2.1 Was ist eine SBOM und welchem Zweck dient sie?

Der Begriff »Bill of Materials« stammt aus der physikalischen Welt, wo Stück- oder Material-Listen für physikalische Güter gepflegt werden. Darüber wird Transparenz über Bestandteile von Produkten geschaffen und Daten wie Bezeichnung, Herkunft oder Hersteller erfasst.

Herstellern gibt diese Bill of Materials zum Beispiel die Möglichkeit, bei Herstellungsproblemen zu erkennen, welche Komponenten betroffen sind, auf welche Produkte sich das auswirkt und durch wen die Ursachen behoben werden können.

145 ↗ »Executive Order on Improving the Nation's Cybersecurity«

Für Konsumentinnen und Konsumenten gibt die Bill of Material Transparenz über die Bestandteile, wie zum Beispiel die Zutatenliste bei Lebensmitteln. Damit kann die Auswahl von Lebensmitteln, etwa bei Allergien, bewusst getroffen werden.

Übertragen auf die Software-Welt beschreibt die Software Bill of Materials die kompletten Bestandteile eines gegebenen Software-Produkts oder einer Software-Komponente. Dabei werden insbesondere Daten erfasst wie eindeutige Identifizierung eines Bestandteils, Versionen, Lizenzen und Informationen über die Struktur wie Abhängigkeitsinformationen.

Insbesondere Abhängigkeiten können eine Herausforderung darstellen, da sie oft nur implizit ausgewählt werden, selbst bei kleinen Produkten eine große Zahl enthalten sein kann und ohne explizite Analyse sich oft überraschende Bestandteile ergeben können.

Sind alle Komponenten über eine SBOM erfasst, ermöglicht das die Frage zu beantworten: »Welche exakte Software setze ich eigentlich ein?« Das bietet die Grundlage, um Aktivitäten durchzuführen, wie:

- **Behandlung von Schwachstellen.** Welche Anwendungen sind betroffen? Wie ist der aktuelle Patch-Status?
- **Lizenz-Compliance.** Welche Lizenzen sind in der Software enthalten? Gibt es Inkompatibilitäten oder Lizenzen mit unerwünschten Auswirkungen?
- **Risiko-Bewertung.** Sind Komponenten enthalten, die Risiken zum Beispiel in Bezug auf Sicherheit oder Support darstellen? Wie häufig werden Komponenten mit Risiken eingesetzt? Worauf wirken die Risiken sich aus?
- **Strategische Übersicht.** Wie sieht das Gesamtbild eingesetzter Software-Komponenten aus? Wie können Software-Komponenten strategisch gestaltet werden? Wo ist zum Beispiel aus strategischer Sicht das Beitragen zu Open-Source-Komponenten wertvoll?

Damit SBOMs ihren Zweck erfüllen können, müssen sie zusammen mit den zugehörigen Software-Artefakten gepflegt und zur Verfügung gestellt werden. Die Verantwortung liegt dabei beim Ersteller des jeweiligen Artefaktes. Das heißt, bei selbst erstellter Software muss eigenständig die Verfügbarkeit von SBOMs sichergestellt werden. Bei zugelieferter Software liegt die Verantwortung bei den Zuliefernden und muss von den Abnehmenden eingefordert werden. Wenn dies über die gesamte Software-Lieferkette sichergestellt wird, kann das Konzept SBOM seine volle Wirkung entfalten.

9.2.2 Welche Daten sind in einer SBOM enthalten?

Eine SBOM enthält einen kompakten Satz von Daten, um Software-Komponenten zu identifizieren sowie die wichtigsten Meta-Daten zu jeder Komponente wie Herkunft oder Lizenz. Dazu enthält eine SBOM Meta-Informationen auf Dokumenten-Ebene wie den Zeitpunkt und die Art der Erstellung und worauf sich die SBOM bezieht.

Es gibt zwei wichtige Standards, um SBOMs zu beschrieben: SPDX und CycloneDX. Sie werden in einem späteren Abschnitt im Detail beschrieben. Das Minimal-Profil von SPDX definiert folgende Daten zur Erfassung (Quelle: ↗ SPDX Spezifikation 2.3):

#	SPDX subclause	Field name
L1.1	6.1	SPDX-Version
L1.2	6.2	Data License
L1.3	6.3	SPDX Identifier
L1.4	6.4	Document Name
L1.5	6.5	SPDX Document Namespace
L1.6	6.8	Creator
L1.7	6.9	Created
L2.1	7.1	Package Name
L2.2	7.2	Package SPDX Identifier
L2.3	7.3	Package Version
L2.4	7.4	Package File Name
L2.5	7.5	Package Supplier
L2.6	7.7	Package Download Location
L2.7	7.8	Files Analyzed
L2.8	7.11	Package Home Page
L2.9	7.13	Concluded License
L2.10	7.15	Declared License
L2.11	7.16	Comments on License
L2.12	7.17	Copyright Text
L2.13	7.20	Package Comment
L2.14	7.21	External Reference Field
L3.1	10.1	License Identifier
L3.2	10.2	Extracted Text

#	SPDX subclause	Field name
L3.3	10.3	License Name
L3.4	10.5	License Comment

Eine SBOM bezieht sich immer auf ein bestimmtes Software-Artefakt. Das kann zum Beispiel ein Software-Binary sein, das zum Deployment in Produktion erzeugt wird oder ein bestimmtes Release eines Open-Source-Projektes.

Wichtig ist, dass die SBOM für das gegebene Artefakt eine vollständige Liste aller enthaltenen Software-Komponenten darstellt. Nur so lassen sich zum Beispiel Fragen zur Sicherheit oder Lizenz-Compliance zuverlässig beantworten.

Open-Source-Komponenten sind ein wichtiger Aspekt von SBOMs. Sie werden in der Anzahl die meisten SBOMs dominieren. Aber es müssen auch proprietäre und selbst entwickelte Komponenten erfasst werden, damit die Vollständigkeit gewährleistet ist und einheitliche Vorgehensweisen und Werkzeuge angewandt werden können.

Je nach Fragestellung kann ein unterschiedlicher Scope von SBOMs notwendig sein. Aus Lizenzsicht ist zum Beispiel die Liste aller ausgelieferten Komponenten oft ausreichend. Aus Sicherheitssicht kann es auch notwendig sein, verwendete Komponenten einer Bau-Umgebung zu erfassen.

Es muss auch abgegrenzt werden, welche Daten in eine SBOM gehören und welche separat davon gehalten werden. Da die SBOM den Stand eines Software-Artefakts widerspiegelt, sollte der Lebenszyklus der SBOM der gleiche sein wie der des Artefaktes. Das heißt, dass statische Informationen, wie Lizenzinformationen in der SBOM erfasst werden sollten, aber dynamische Informationen wie welche Schwachstellen in den Komponenten enthalten sind, besser separat gepflegt werden.

9.2.3 Wie wird mit Unvollständigkeit umgegangen, wie wird das dokumentiert?

Es kann in manchen Fällen schwierig, unmöglich oder wirtschaftlich nicht sinnvoll sein, eine vollständige Erfassung durchzuführen. Dies sollte entsprechend dokumentiert sein, so dass auch die »Known Unknowns« erfasst sind. Im SPDX-Format gibt es dazu zum Beispiel die Möglichkeit, Datenfelder als »NOASSERTION« zu kennzeichnen, um transparent zu machen, dass keine Lizenz-Informationen erfasst wurden (was nicht das gleiche ist wie eine Komponente ohne Lizenz).

9.2.4 Wie wird eine SBOM erzeugt?

Software enthält in vielen Fällen eine sehr große Zahl von Komponenten. Selbst einfache Javascript-Anwendungen kommen oft auf über 10.000 Abhängigkeiten und auch in konservativeren Ökosystemen wie Java bestehen Anwendungen oft aus Hunderten von Komponenten.

Diese Zahl kann nur durch Automatisierung wirtschaftlich behandelt werden. Im Einzelfall mag auch eine manuell erstellte SBOM ihren Zweck erfüllen, aber eine vollständige Erfassung ist nur durch den Einsatz von entsprechenden Werkzeugen möglich.

Es gibt eine sehr aktive Entwicklung im Bereich SBOM-Tools, sowohl durch Hersteller proprietärer Angebote als auch in der Open-Source-Community.

Je nach Einsatzszenario, Sprach-Ökosystem und konkreter Software-Produkte bieten sich verschiedene Lösungen an. Es ist zu empfehlen, pragmatische Lösungen zu suchen, die auf die Bedürfnisse von Entwicklungsteams eingehen, um zu gewährleisten, dass SBOMs auch tatsächlich erzeugt werden und der notwendige Aufwand dafür sich in Grenzen hält. Wegen der Dynamik des Marktes und der rapiden Weiterentwicklung des Themas ist ebenso zu empfehlen, Flexibilität hinsichtlich der Werkzeuge zu erhalten und den Fokus auf Standard-Formate und APIs zu legen. Mehr dazu im Abschnitt ↗ Tools zur Verwaltung von SBOMs.

Bestehende und insbesondere ältere Software kann hier bei der SBOM-Erstellung eine Herausforderung darstellen. Bei modernen Produktions-Modellen, wie zum Beispiel DevOps, ist es recht einfach, entsprechende Tools in vorhandene Automatisierung einzubauen. Bei anderen Modellen kann das schwieriger sein.

9.2.5 Wie wird eine SBOM übermittelt?

SBOMs werden als Teil des Software-Herstellungsprozesses erzeugt. Damit sie fürSoftware-Konsumentinnen und -Konsumenten nutzbar werden, müssen sie diesem auf geeignete Weise bereitgestellt werden. Hierfür gibt es über die Daten-Formate hinaus noch wenig Standardisierung. Um die Integrität der SBOMs zu garantieren, ist es empfehlenswert, diese durch digitale Signaturen zu schützen und Signaturen bei der Verwendung entsprechend zu überprüfen[146].

146 Dies kann zum Beispiel durch Nutzung von ↗ Tools des SigStore-Projektes geschehen.

9.2.6 Wie wird eine Metrik für die Zuverlässigkeit und Vertrauenswürdigkeit einer 3rd Party-Zulieferung einer SBOM definiert?

Es besteht immer das Risiko, dass eine SBOM unvollständig oder fehlerhaft ist – sei es durch Unzulänglichkeiten oder Probleme bei der SBOM-Erstellung als auch durch gezielte Manipulation. Deshalb ist es notwendig, bei zugelieferten SBOMs ihre Vertrauenswürdigkeitzu bewerten und bei selbst erstellten SBOMs ihre Zuverlässigkeit sicher zu stellen.

9.2.7 Welche Standards und Formate kommen bei SBOMs zur Anwendung?

Es haben sich im Wesentlichen zwei Standards für SBOMs etabliert. Der von der Linux Foundation unterstützte ↗ SPDX-Standard, der seit 2021 auch als ISO-Standard ISO/IEC 5962:2021 anerkannt ist sowie ↗ CycloneDX, das aus der ↗ OWASP-Community stammt. Daneben gibt es noch ↗ SWID, welches allerdings im Bereich SBOM eine eher untergeordnete Bedeutung hat.

Neben dem Datenmodell definieren die Standards verschiedene Austauschformate, wie JSON oder RDF, so dass effiziente maschinelle Verarbeitung und Austausch von SBOMs ermöglicht wird. Auch zur Konvertierung zwischen verschiedenen Formaten existieren Tools.

Neben den Standards für die SBOM als Dokument sind auch noch Standards für einzelne Elemente relevant. Hier ist zum einen ↗ SPDX Id zu nennen, womit eindeutige Kennzeichnungen für Open-Source-Lizenzen geschaffen werden. Zum anderen gibt es noch ↗ package URL, mit dem Software-Pakete eindeutig identifiziert werden können. Beide Standards kommen in SPDX und CycloneDX zur Anwendung.

9.2.8 Wie werden SBOMs im Unternehmen verwaltet?

SBOMs müssen im Kontext der zugehörigen Software-Artefakte verwaltet werden. Ihren Nutzen entfalten sie dabei aus zwei Perspektiven:

Für ein einzelnes Software-Artefakt bieten sie eine Möglichkeit, dessen Zusammensetzung und Herkunft transparent zu machen. Damit können spezifische Fragen zu einer bestimmten Software beantwortet werden. Das kann nützlich für ein Entwicklungs- oder Betriebs-Team sein, das mit dieser Software umgeht.

Die Aggregation von SBOMs für alle eingesetzte Software bietet zusätzliche Möglichkeiten, zentrale Fragen zu beantworten – zum Beispiel, wie stark ein Unternehmen von einer bestimmten Software-Schwachstelle betroffen ist oder welche Abhängigkeiten zu welcher Software bestehen. Es lassen sich auf diese Weise auch Prüfungs-Prozesse, zum Beispiel zum Thema Lizenz-Compliance, umsetzen und automatisieren.

Wie SBOMs im Unternehmen verwaltet werden, so dass sie ihren Nutzen entfalten können, hängt stark vom technischen Umfeld, Prozessen und der Organisation eines Unternehmens ab. Dabei spielen folgende Themen eine Rolle:

- Einbindung in CI/CD-Infrastruktur
- Anbindung ans Asset-Management
- Vorgaben und Handlungsempfehlungen, wann SBOMs in welcher Form erzeugt oder beschafft werden müssen
- Integration in ITIL
- Lifecycle-Management
- Standards und Schnittstellen, um SBOMs ablegen und auf die in ihnen enthaltenen Informationen zugreifen zu können

9.2.9 Analysieren und Visualisieren von SBOMs

Durch das maschinenlesbare Format von SBOMs ist es möglich, die darin enthaltenen Informationen effektiv zu analysieren oder zu visualisieren. Dies kann durch generische Daten-Management-Tools geschehen, die zum Beispiel auf dem JSON-Format arbeiten, durch spezifische Skripte oder durch spezialisierte Tools zur SBOM-Analyse.

9.2.10 Tools zur Verwaltung von SBOMs

Es gibt eine Reihe von Tools, die bei der Verwaltung von SBOMs zum Einsatz kommen können. Der Markt ist stark in Bewegung und sowohl im Bereich proprietärer Tools als auch im Bereich von Open-Source-Tools findet viel Entwicklung statt.

Es kann keine eindeutige Empfehlung abgegeben werden, welche Tools ein Unternehmen einsetzen sollte. Das muss im jeweiligen Umfeld analysiert und entschieden werden. Eine Hilfestellung dabei kann die Klassifizierung von Tools nach verschiedenen Einsatzfeldern sein (Quelle: ↗ »SBOM Tool Classification Taxonomy«, NTIA SBOM Formats & Tooling Working Group):

Eine Tool-Übersicht aus der Sicht der beiden SBOM-Standardformate findet sich unter ↗ SPDX-Tools und ↗ CycloneDX-Tools.

9.2.11 Was ist bei der Beschaffung von Software von Zulieferern in punkto SBOM zu beachten?

Zugelieferte Software stellt oft eine »Black Box« dar, bei der nicht erkennbar ist, wie sie entwickelt wurde, welche Komponenten zum Einsatz kommen und woher diese Komponenten stammen. Open-Source-Lizenzen erfordern in der Regel, dass zumindest Lizenz-Texte mit ausgeliefert werden. Dies lässt aber meist nicht den Rückschluss auf die exakte Zusammensetzung und Versionen von Komponenten zu. Eine SBOM kann diese Lücke füllen.

Es ist noch nicht selbstverständlich, dass Software mit einer SBOM als »Zutatenliste« kommt. Manche Software-Hersteller bewegen sich in diese Richtung und stellen SBOMs oder vergleichbare Informationen von sich aus zur Verfügung. Damit dies flächendeckend geschieht, müssen Software-Abnehmer dies jedoch in der Breite einfordern.

In manchen Bereichen geschieht dies durch allgemeine Vorgaben, wie zum Beispiel die US-amerikanische Executive Order. Sie wird es verpflichtend machen, für Software, die an öffentliche Stellen geliefert wird, SBOMs mitzuliefern.

In der Regel wird es aber der individuellen Gestaltung durch Unternehmen überlassen sein, wie die Verfügbarkeit von SBOMs durchgesetzt werden kann. Dies kann zum Beispiel in Verträgen mit Zulieferern standardmäßig gefordert werden.

Bei Open-Source-Software ist es durch die Verfügbarkeit des Quellcodes oft möglich, SBOMs selbst zu erstellen, so dass Angaben von Herstellern überprüft oder fehlende Informationen ergänzt werden können.

Ein Standard-Vorgehen wird in der ↗ OpenChain-Spezifikation definiert, die den Umgang mit Open-Source-Software in der Lieferkette beschreibt. Sie ist seit 2021 auch als Standard ISO/IEC 5230 verfügbar.

9.2.12 Was ist bei der Auslieferung von Software an Dritte in punkto SBOM zu beachten?

Wenn SBOMs bei der Zulieferung von Software verpflichtend gefordert werden, stellt das eine Anforderung an Software-Hersteller dar. Diese müssen die Erzeugung und Auslieferung von SBOMs in ihre Prozesse zur Erstellung und Auslieferung von Software integrieren. Um vollständige und genaue Informationen gewährleisten zu können, ist es unerlässlich, diese Prozesse zu automatisieren.

9.2.13 Einordnung der Initiativen zu SBOMs: Was geschieht in Amerika, was geschieht in Europa und speziell in Deutschland?

Die eingangs erwähnte präsidiale Verordnung 14208 hat in den USA eine starke Beschäftigung mit dem Thema SBOM ausgelöst. Die Perspektive, dass Software, die in der öffentlichen Verwaltung eingesetzt wird, zwingend mit einer SBOM kommen muss, setzt Software-Hersteller unter Druck. Das ist eine Herausforderung und gleichzeitig eine Chance, Prozesse, Vorgehensweisen und Werkzeuge zum Umgang mit SBOMs zu entwickeln und zu einem hohen Reifegrad zu bringen. Das sorgt füreine starke Dynamik im amerikanischen Markt.

Auch in der Open-Source-Community ist das Thema sehr präsent. Projekte wir Kubernetes haben inzwischen die Erzeugung von SBOMs zum Teil ihres Release-Prozesses gemacht (Siehe ↗ »We Built the Kubernetes SBOM and Now You Can Write Your Own!«, Adolfo García Veytia), es gibt eine wachsende Zahl von Open-Source-Tools zum Umgang mit SBOMs und auch in Initiativen, wie zum Beispiel der 2020 gegründeten ↗ Open Source Security Foundation (OpenSSF), wird das Thema behandelt.

Eine Initiative, die eine sehr starke Industrie-Unterstützung hat, ist der von OpenSSF und einigen Dutzend Technologie-Firmen unterstützte ↗ »The Open Source Software Security Mobilization Plan«. Im Rahmen dieses Plans werden 150 Millionen US-Dollar bereitgestellt, um die Sicherheit der Software-Lieferkette zu erhöhen. SBOM ist einer der 10 »Streams« des Planes.

Auf europäischer Ebene gibt es die Network and Information Security (NIS)-Direktive, die der erste Ansatz ist, Cybersecurity-Standards auf europäischer Ebene zu setzen. Eine aktualisierte Version (↗ NIS2) ist gerade in Abstimmung. Sie fügt eine Förderung von Open-Source-Tools zur Cybersicherheit hinzu, um insbesondere auch kleinen und mittleren Unternehmen den Zugang zu adäquaten Werkzeugen zu ermöglichen. Die Direktive spricht das Thema SBOM allerdings nicht an.

Eine Initiative der EU ist der ↗ Cyber Resilience Act, der im Entwurf vorliegt. Er adressiert direkt die Sicherheit von digitalen Produkten, indem Hersteller zu sichernden

Maßnahmen verpflichtet werden sollen und Konsumentinnen und Konsumenten Transparenz über sicherheitsrelevante Aspekte von Produkten bekommen. In dem Entwurf ist die SBOM als ein Mittel vorgesehen, um dies zu erreichen. Ein interessanter Aspekt des Entwurfs ist, dass er Open-Source-Software, die nicht in einem kommerziellen Kontext entwickelt wird, von vielen Verpflichtungen ausnimmt, die im kommerziellen Kontext gefordert werden.

9.2.14 Ausblick: Wie wird sich das Thema SBOM entwickeln?

Das Thema SBOM hat in den letzten Jahren erheblich an Fahrt aufgenommen. Es wird breit diskutiert und es gibt eine Vielzahl von Lösungsvorschlägen für verschiedenste Aspekte. Dabei ist vieles noch in einem frühen Stadium der Entwicklung und Vorgehensweisen müssen noch erprobt werden und reifen. Es gibt aber schon einige Beispiele, wo SBOMs belastbar eingesetzt werden.

Eine SBOM an sich löst noch kein Problem, aber sie schafft die Grundlage, um mit standardisierten und automatisierten Methoden einen Mehrwert zu schaffen. Dies können die Erkennung und Beseitigung von Schwachstellen sein, die Analyse und Quantifizierung von Risiken, die aus der Software-Lieferkette entstehen oder Bewertung und Bearbeitung von Compliance-Aspekten, wie zum Beispiel zu Open-Source-Lizenzen.

Da hier das Zusammenspiel von all den Open-Source-Projekten, Herstellern und Konsumentinnen und Konsumenten nötig ist, die an Software-Lieferketten beteiligt sind, scheint es unausweichlich zu sein, dass sich Standards durchsetzen, die Interoperabilität verschiedener Werkzeuge und SBOMs aus verschiedenen Quellen gewährleisten. Die Community und Industrie sind gut beraten, in diese Richtung zu arbeiten.

Aus der Vielzahl von Open-Source-Komponenten ergibt sich ein starker Druck zur Automatisierung. Das Konzept SBOM wird nur dann erfolgreich sein, wenn es keine große Last bei Entwicklung, Auslieferung und Einsatz von Software darstellt. Dies kann durch Automatisierung und gute Integration über Standard-Formate und Schnittstellen gelingen.

9.2.15 Referenzen

- ↗ SBOM-Seite der NTIA – Übersichtsseite der US-amerikanischen National Telecommunications and Information Administration zum Thema SBOM mit recht tiefgehenden Grundlagenmaterial
- ↗ SBOM-Seite der CISA – Übersichtsseite der US-amerikanischen Cybersecurity & Infrastructure Agency mit Informationen zur Umsetzung des SBOM-Konzepts
- ↗ The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness – Report der Linux Foundation zum Stand des Themas SBOM in der Industrie
- ↗ Finding vulnerabilities with a SBOM – Beispiel, wie SBOMs verwendet werden können, um Schwachstellen zu finden
- ↗ Awesome SBOM – Kuratierte Liste von Materialien zum Thema SBOM
- ↗ Software Component Transparency: Healthcare Proof of Concept Report – Eins der frühen Dokumente, aus dessen Umfeld die Idee der SBOM stammt

Anhang

Abkürzungsverzeichnis

AGB
Allgemeine Geschäftsbedingungen

ASP
Application Service Providing

BGB
Bürgerliches Gesetzbuch

BHO
Bundeshaushaltsordnung

CPU
Central Processing Unit

FAQ
Frequently Asked Questions

FOSS
Freie und Open-Source-Software

Gem HVO NRW
Verordnung über das Haushaltswesen der Gemeinden
im Land Nordrhein-Westfalen

GWB
Gesetz gegen Wettbewerbsbeschränkungen

HGrG
Gesetz über die Grundsätze des Haushaltsrechts des Bundes und der Länder

LHO NRW
Landeshaushaltsordnung des Landes Nordrhein-Westfalen

LTS
Long Term Support

OSD
Open-Source-Definition der OSI

OSI
Open-Source-Initiative

OSS
Open-Source-Software

PatG
Patentgesetz

SäHO
Haushaltsordnung des Freistaates Sachsen

SBOM
Software Bill of Materials

SPDX
Software Package Data eXchange

UrhG
Gesetz über Urheberrecht und verwandte Schutzrechte

UWG
Gesetz gegen den unlauteren Wettbewerb

VOL/A
Vergabe- und Vertragsordnung für Leistungen – Teil A für Vergaben öffentlicher Auftraggeber bei Liefer- und Dienstleistungsaufträgen

Literatur- und Quellenverzeichnis

[1] Benkard, Georg u. a.:
Patentgesetz, Gebrauchsmustergesetz, Patentkostengesetz; Kommentar;
10. Aufl.; Verlag C. H. Beck, München 2006

[2] Gerlach, Carsten:
Vergaberechtsprobleme bei der Verwendung von Open-Source-Fremdkomponenten, in Computer und Recht 2012 (Heft 10), S. 691–696

[3] Heiermann, Wolfgang / Zeiss, Christopher (Hrsg.):
juris PraxisKommentar Vergaberecht, 4. Auflage, Verlag Juris Saarbrücken 2013

[4] Institut für Rechtsfragen der Freien und Open-Source-Software (verschiedene Autoren):
Die GPL kommentiert und erklärt, 1. Auflage März 2005 (im Internet unter:
http://www.oreilly. de/german/freebooks/gplger/)

[5] Jaeger, Till / Metzger, Axel:
Open-Source-Software – Rechtliche Rahmenbedingungen der Freien Software,
3. Auflage, Verlag C. H. Beck, München 2011

[6] Lamon, Bernard: Le droit des licences Open Source, Version 1.1 (August 2009);
(im Internet unter: http://www.bernardlamon.fr/wp-content/uploads/2009/07/
livre-blanc-v3-aout-2009.pdf)

[7] Laurent, Phillippe:
»Open-Source-/Content Licenses before European Courts«, EOLE 2012;
(im Internet unter: http://www.eolevent.eu/en/speeches2012)

[8] MPEP:
Manual of Patent Examining Procedure, Handbuch für die Prüfer des US-amerikanischen Patentamts, (im Internet unter: www.uspto.gov)

[9] Picot, Henriette:
»Die deutsche Rechtsprechung zur GNU General Public License«, in: Open-Source-Jahrbuch 2008, S. 184 ff.

[10] Reincke, Karsten / Sharpe, Greg:
Open-Source-License Compendium – How to Achieve Open-Source-License Compliance, Darmstadt, Bonn 2015; (im Internet unter: http://openSource.telekom.
net/oslic/releases/oslic.pdf)

[11] Schöttle, Hendrik:
Der Patentleft-Effekt der GPLv3, in: Computer und Recht 1/2013, S. 1 ff.

[12] Van den Brande, ein / Coughlan, Shane / Jaeger, Till (Hrsg.):
The International Free and Open-Source-Software Law Book, 2nd edition 2014;
(im Internet unter: http://ifosslawbook.org/)

[13] Working Group Public Affairs der OSB Alliance (Autor: Till Jaeger):
Handreichungen zur Nutzung der EVB-IT beim Einsatz von Open-Source-Software,
2013 (im Internet unter: http://osb-alliance.de/fileadmin/Downloads/EVB-IT_
Handreichungen/ Positionspapier_EVB-IT_170314.pdf)

[14] Wuermeling, Ulrich / Deike, Thies:
»Open-Source-Software: Eine juristische Risikoanalyse«; in: Computer und
Recht 2/2003, S. 87 ff.

Literaturergänzungen

[1] Cluster Mechatronik & Automation e.V., Open-Source-Software, Leitfaden
zum Einsatz in Unternehmen, 2. erweiterte Ausgabe, 2014 (im Internet unter:
http://www.cluster-ma.de/ publikationen/leitfaden-oss-2-erweiterte-ausgabe/
index.html)

[2] Susanne Strahringer (Hrsg.): Open Source – Konzepte, Risiken, Trends, Praxis
der Wirtschaftsinformatik, HMD 283, 2012, dpunkt.verlag

[3] International FOSS law review: http://www.ifosslr.org/ifosslr (Trotz fehlender
Neupublikationen seit 2014 bietet diese Site einen reichen Fundus an rechtlichen
Informationen zu Open-Source-Software.)

Glossar

Branch

Bei den für die Softwareentwicklung typischerweise verwendeten Versionskontrollsystemen wird die zur Strukturierung des Quelltexts eines Entwicklungsprojekts verwendete Verzeichnisstruktur als Baum bezeichnet. Ein Teilprojekt oder ein Task zweigt dann von diesem Baum ab und ermöglicht so die vom restlichen Projekt weitgehend unabhängige Entwicklung von Features oder Fixes im eigenen Zweig. Dieser Zweig wird als Branch bezeichnet. Bei Abschluss einer jeweils abgezweigten Entwicklung werden die Ergebnisse dann durch einen Merge mit einem anderen Zweig oder mit dem Hauptast vereinigt.

Code-of-Conduct

Ein Code-of-Conduct ist ein Verhaltenskodex bzw. eine Verhaltensrichtlinie. In diesem Dokument werden Erwartungen an das Verhalten von Teilnehmenden in einer Community festgelegt.

Community

Im Zusammenhang mit Open Source ist eine Community eine Gruppe von Menschen im Umfeld eines Open-Source-Projekts. Die Community nutzt eine bestimmte Open-Source-Software, beteiligt sich an der Entwicklung und kann das Projekt auch anderweitig unterstützen.

Contribution

Unter einer Contribution versteht man den inhaltlichen Beitrag einer Person oder Gruppe zu einem Open-Source-Projekt. Das sind zum Beispiel Quelltexte, Dokumentation oder Grafiken.

Contributor License Agreement

Ein Contributor License Agreement (CLA) ist eine rechtliche Vereinbarung in der die Bedingungen für die Contribution festgelegt werden.

Copyleft

Das Copyleft ist eine Bedingung von einigen Open-Source-Lizenzen. Gilt für eine Software eine Open-Source-Lizenz mit Copyleft-Bedingung, muss diese Software, Veränderungen der Software und von der Software abgeleitete Werke grundsätzlich unter derselben Open-Source-Lizenz weitergegeben werden. Beispielsweise enthält die GNU General Public License in der Version 2 (GPL-2.0) eine Copyleft-Bedingung. Das Copyleft gibt es in verschiedenen Ausprägungen. So kann je nach Ausgestaltung das Copyleft z.B. nur Änderungen an der Software selbst betreffen (sog. schwaches Copyleft) oder sich auch auf die in einem abgeleiteten Werk eigene Softwareentwicklung erstrecken (sog. starkes Copyleft).

(Siehe Kapitel 7.2 Lizenztypen und Compliance-Aktivitäten)

Defacto-Standards

Defacto-Standards sind Standards die sich aufgrund ihrer weit verbreiteten Nutzung und Akzeptanz etabliert haben. Im Gegensatz hierzu stehen Dejure-Standards die durch anerkannte Standardisierungsorganisation entwickelt und verabschiedet werden.

Distribution / Distribuieren / Distributor

Der Begriff Distribution bezeichnet die Verteilung von Software und zugehöriger Artefakte. Dies umfasst meist installierbare Pakete mit ausführbaren Dateien und Dokumentation, kann sich aber auch auf Quellcode, Konfiguration und ähnliches erstrecken. In der Regel wird der Begriff im Sinne von Distribution an Endnutzer und Endnutzerinnen verwendet. Dies kann kostenlos durch eine Community erfolgen oder als kommerzielles, oft mit weiteren Service-Leistungen verbundenes, Angebot eines Herstellers.

Die verteilende Partei ist der Distributor, der Vorgang der Distribution ist das Distribuieren.

Je nach Zusammenhang kann der Begriff auch für die Gesamtheit der distribuierten Artefakte stehen, zum Beispiel in der Verwendung im Begriff »Linux-Distribution«.

Downstream

Die Begriffe Upstream und Downstream werden in Bezug auf die Entwicklung und Verbreitung von Open-Source-Software verwendet.

Downstream bezieht sich auf Zweige oder Ableitungen einer Software, die ursprünglich von Dritten erstellt wurden. Beispiele hierfür sind aus der Software für die Distribution generierte Pakete oder auch ↗Forks und angepasste Versionen einer Software. Ein Downstream-Projekt nutzt dabei den Code eines Upstream-Projekts als Ausgangspunkt für eigene Entwicklungen. Das ist zum Beispiel ein aus dem Upstream-Projekt abgeleitetes kommerzielles Produkt.

Fork

Ein Fork bezeichnet die Ableitungen eines Softwareprojekts in Folgeprojekte, die sich alternativ entwickeln können. Das Original-Projekt, von aus dem der Fork erstellt wird, ist das ↗upstream-Projekt und der Fork das ↗downstream-Projekt. Ein Fork kann sich auch inkompatibel zum Original-Projekt entwickeln und somit unabhängig werden. Auf der Plattform GitHub wird der Begriff Fork in der Regel nur für eine temporäre, technische Ableitung des Original-Projektes verwendet, um eine Änderung einzureichen, und nicht für eine unabhängige Entwicklung. Beispielsweise zum ↗Merge eines ↗»Feature Branch« über einen sogenannten Pull Request.

Foundation (Open Source)

Eine Foundation im Kontext von Open Source ist eine nicht gewinnorientierte Organisation, die darauf abzielt, Open-Source-Projekte zu unterstützen und zu fördern. Sie bietet rechtliche, finanzielle, organisatorische, technische und inhaltliche Unterstützung, um die Entwicklung und Nachhaltigkeit von Open-Source-Software zu gewährleisten. Foundations helfen dabei, die Interessen der Gemeinschaft zu schützen, die Zusammenarbeit zu erleichtern, Projekte gegenüber der Öffentlichkeit und Interessengruppen zu vertreten und auch Standards zu setzen.

Governance

Governance in Open Source bezieht sich auf die Rahmenbedingungen, Regeln, (bewährte) Praktiken und Prozesse, die die Entwicklung, Nutzung und Verwaltung von Open-Source-Software-Projekten steuern. Ziel ist es, die Qualität, Sicherheit und Nachhaltigkeit der Software zu gewährleisten und gleichzeitig die Zusammenarbeit, Verteilung von Verantwortlichkeiten, Mitwirkung und Beteiligung der Community zu fördern. Die Governance bestimmt auch, nach welchen Prinzipien Entscheidungen getroffen werden, wer sie trifft und wie Beiträge verwaltet werden.

Inbound Open Source

Bezeichnet den Ansatz, Open-Source-Software für die Erstellung und/oder als Bestandteil von eigenen Produkten oder Services zu verwenden. Unternehmen und andere Organisationen haben damit die Möglichkeit, Software aus dem Open-Source-Bereich wiederzuverwenden, anstatt eigene Lösungen entwickeln zu müssen. Inbound bezeichnet die Flussrichtung der Software von außerhalb der Organisation nach innen. Die Gegenrichtung nennt man ↗Outbound Open Source.

Merge

Ein Merge bezeichnet das Zusammenführen von Änderungen am Quellcode in Software-Projekten aus z. B. mehreren ↗Branches oder aus einem ↗upstream-Projekt. Technisch wird dies durch ↗Versionskontrollsysteme ermöglicht und durch Pull Requests (GitHub) beziehungsweise Merge Requests (GitLab) unterstützt, die die Möglichkeit bieten, den Quellcode vor dem Merge zu prüfen.

Open-Source-Lizenz

Open-Source-Lizenzen stellen eine Kategorie von Softwarelizenzen dar. Diese erlauben es jedem, die Software auszuführen, zu analysieren, anzupassen und weiterzugeben (die sogenannten vier Freiheiten). Grundsätzlich schließt das den Zugriff auf den Quellcode ein. Die Verwendung darf nicht an die Zahlung einer Lizenzgebühr gebunden sein. Niemand darf von der Nutzung der Software

ausgeschlossen sein und die Nutzung muss für jeden Zweck möglich sein. Bekannte Open-Source-Lizenzen sind u.a. die GNU General Public License (GPL), die Apache v2.0 License und die MIT License. Die ↗ Open Source Initiative (OSI) pflegt eine Liste der Open-Source-Lizenzen.

(Siehe Kapitel 2.2: Konzept und Definition von Open-Source-Software)

Open-Source-Projekt (Community, Technik)
Ein Open-Source-Projekt beschreibt sowohl die Software selbst als auch die Community, die diese entwickelt und pflegt. Im Gegensatz zu proprietärer Software ist der Quellcode grundsätzlich offen zugänglich und kann von jedermann eingesehen, bearbeitet und weiterverbreitet werden. Diese Offenheit ermöglicht eine kollaborative Entwicklung, an der sich jeder mit Fähigkeiten und Interesse beteiligen kann. Die Leitprinzipien dieser Zusammenarbeit werden durch eine Open-Source-Lizenz definiert, die die Rechte und Pflichten der Nutzer festlegt.

Outbound Open Source
Bezeichnet den Ansatz, zu einem Open-Source-Projekt beizutragen (siehe ↗Contribution) beziehungsweise eigene Open-Source-Projekte zu initiieren. Unternehmen und andere Organisationen können durch Contributions u.a. fehlende Funktionalitäten in Open-Source-Projekten ergänzen, die sie verwenden (siehe ↗Inbound Open Source). Eigene Open-Source-Projekte zu starten, d. h. Software unter einer Open-Source-Lizenz zu veröffentlichen, ermöglicht die Zusammenarbeit über Unternehmen und Organisationen hinweg, fördert Innovationen und stärkt das Vertrauen in die Software. Outbound bezeichnet die Flussrichtung der Software aus der Organisation heraus.

Patches (als Contribution)
Der sinnbildliche Flicken zur Reparatur oder Ausbesserung eines Fehlers oder Mangels bezeichnet eine inhaltlich abgeschlossene Änderung an einer Software. Im Kontext von Open-Source-Entwicklungen werden solche Änderungen von einer Person oder Gruppe erstellt und in dem öffentlichen Entwicklungsprozess praktisch von außen als Änderungsvorschlag an das Projekt herangetragen. Indem eine Änderung über den ↗Merge-Prozess in das Open-Source-Projekt übernommen wird, gelangt der Patch als ↗Contribution in das [Upstream](#upstream- Projekt.

Proprietär
Der Begriff proprietär wird in Abgrenzung zu Open Source für Software verwendet, bei der sich der Rechteinhaber wesentliche Rechte vorbehält und Nutzerinnen und Nutzern nicht die vollen Freiheiten wie bei Open-Source-Software gewährt. So ist der Quellcode proprietärer Software üblicherweise nicht öffentlich zugänglich.

Release
Als Release bezeichnet man einen spezifischen Entwicklungsstand einer Software, der von dem entwickelnden Team Nutzern zur Verfügung gestellt wird. Dabei werden meist zusätzliche Release-Artefakte erstellt, wie herunterladbare Archive, die den entsprechenden Stand der Software enthalten, und Release-Notes mit Dokumentation der Änderungen im Vergleich zu früheren Releases.

Releases werden in der Regel mit Versionsnummern oder ähnlichen Bezeichnungen gekennzeichnet, die zur exakten Identifikation bestimmter Releases genutzt werden und auch eine zeitliche Einordnung von verschiedenen Releases oder des Umfangs der Änderungen zwischen Releases ermöglichen.

Repository
Repository ist der Fachbegriff für den Ort, wo der Code eines Software-Projekts verwaltet wird. Meist bezieht sich dieser Begriff auf den Quellcode, aber es gibt auch Repositories von anderen Artefakten, wie zum Beispiel installierbaren Software-Paketen. Der Begriff stammt aus dem Bereich der Versionskontrollsysteme.

Software Bill of Materials

Eine Software Bill of Materials (SBOM) enthält Einträge zu allen Bestandteilen bzw. Komponenten einer Software. In den Einträgen sind die eindeutige Bezeichnung und die Version der Softwarebestandteile festgehalten. Als weitere Information befindet sich in den Einträgen der SBOM die Lizenz, die für den Softwarebestandteil gilt. Für die Formatierung und den Austausch von SBOMs gibt es mehrere Standards.

(Siehe Kapitel 9.2: Software Bill of Materials (SBOM))

Software-Komponente

Software ist üblicherweise aus einer Vielzahl von Komponenten aufgebaut, die in der Regel aus verschiedenen Quellen kommen und unabhängig voneinander entwickelt werden. In diesem Leitfaden verwenden wir dafür durchgängig den Begriff Software-Komponente. Alternativ ist dafür auch der Begriff Software-Modul üblich, insbesondere in manchen Programmiersprachen-Ökosystemen.

Ein großer Teil dieser Software-Komponenten sind Bibliotheken, also Sammlungen von Funktionen, die Programmierer und Programmiererinnen bei der Software-Entwicklung nutzen können. Es gibt aber auch anders geartete Komponenten, die Werkzeuge bereitstellen, die zur Laufzeit eingebunden werden, etc.

Software-Komponenten müssen nicht notwendigerweise Open Source sein, bei aktueller Software-Entwicklung steht aber der mit Abstand größte Anteil von verwendeten Komponenten unter einer Open-Source-Lizenz.

Software-Lieferkette (Software-Supply-Chain)

Eine Software-Lieferkette (eng. Software Supply Chain) [↗1] besteht aus den Komponenten, Tools und Prozessen sowie Organisationen, die zur Entwicklung, Erstellung und Veröffentlichung eines Software-Artefakts verwendet werden bzw. daran beteiligt sind. Siehe auch ↗Software Bill of Materials.

Software-Paket

Ein Software-Paket ist eine Software-Komponente, die in einer einfach nutzbaren Form distribuiert wird. Üblicherweise geschieht das unter Verwendung eines Paketmanagers, der Teil des Betriebssystems ist, oder als Standardwerkzeug eines Sprach-Ökosystems zur Verfügung steht. Das Erstellen eines Paketes, das sogenannte Paketieren, ist oft eine Kerntätigkeit eines Distributors.

Upstream

Die Begriffe Upstream und Downstream werden in Bezug auf die Entwicklung und Verbreitung von Open-Source-Software verwendet.

Upstream bezieht sich auf den Hauptentwicklungszweig oder die Quelle einer Software, die von den ursprünglichen Entwicklern oder der offiziellen Projektgemeinschaft verwaltet wird. Neue Funktionen, Fehlerbehebungen und weitere Änderungen werden oftmals im Upstream-Zweig vorgenommen und können von Downstream-Projekten übernommen werden.

Versionskontrollsystem

Mit Versionskontrollsystem bezeichnet man das System, mit dem der Quellcode einer Software verwaltet wird. Dabei werden alle Änderungen in spezifischen Versionen erfasst, einschließlich Informationen über Urheber und Zeitpunkt der Änderungen, so dass sich die Historie eines Software-Projekts lückenlos verfolgen lässt. Zudem ermöglichen Versionskontrollsysteme die kontrollierte Zusammenarbeit von mehreren Personen am gleichen Software-Projekt. So können verschiedene Zweige der Entwicklung in sogenannten Branches abgebildet werden und Werkzeuge zur Verfügung stehen, um Branches wieder zusammenzuführen und dabei möglicherweise entstehende Konflikte aufzulösen.

Das aktuell populärste Versionskontrollsystem ist ↗git und darauf aufbauenden Plattformen wie ↗GitHub und ↗GitLab.

Footnotes

1. ↗Software-Lieferkette bei Wikipedia

Stichwortverzeichnis

Bitkom vertritt mehr als 2.000 Mitgliedsunternehmen aus der digitalen Wirtschaft. Sie erzielen allein mit IT- und Telekommunikationsleistungen jährlich Umsätze von 190 Milliarden Euro, darunter Exporte in Höhe von 50 Milliarden Euro. Die Bitkom-Mitglieder beschäftigen in Deutschland mehr als 2 Millionen Mitarbeiterinnen und Mitarbeiter. Zu den Mitgliedern zählen mehr als 1.000 Mittelständler, über 500 Startups und nahezu alle Global Player. Sie bieten Software, IT-Services, Telekommunikations- oder Internetdienste an, stellen Geräte und Bauteile her, sind im Bereich der digitalen Medien tätig oder in anderer Weise Teil der digitalen Wirtschaft. 80 Prozent der Unternehmen haben ihren Hauptsitz in Deutschland, jeweils 8 Prozent kommen aus Europa und den USA, 4 Prozent aus anderen Regionen. Bitkom fördert und treibt die digitale Transformation der deutschen Wirtschaft und setzt sich für eine breite gesellschaftliche Teilhabe an den digitalen Entwicklungen ein. Ziel ist es, Deutschland zu einem weltweit führenden Digitalstandort zu machen.

Bitkom e.V.

Albrechtstraße 10
10117 Berlin
T 030 27576-0
bitkom@bitkom.org

bitkom.org

bitkom