bitkom

Open Source Guide

Practical recommendations for
Open Source Software
Version 3.1

In the beginning,
all software was free.

Georg C. F. Greve

Publisher
Bitkom e. V.
Albrechtstraße 10 | 10117 Berlin

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

Bitkom Working Group
AK Open Source

Project management
Sebastian Hetze | Red Hat GmbH

Design
Anna Stolz

Picture credits
© kkolis – stock.adobe.com

Inhalt

Foreword

This guide to Open Source Software (OSS) was developed by the Open Source Working Group. This working group brings together experts from Bitkom member companies who, in their professional activities, are intensively involved with the use of Open Source Software in a professional corporate environment. However, the creation of the guide is largely based on their personal voluntary commitment. In the past months, in some cases years (the first ideas for revising the existing guide were already collected and exchanged in February 2020), all those involved have dedicated themselves to the project of revising the guide with a very high level of commitment, great attention to detail and with the utmost professionalism.

In the process, the best structure, the best content and the best formulations were fought over with a lot of passion and in countless telephone and video conferences, in issues and pull requests on GitHub as well as in emails and messages, points of view, perspectives and opinions were exchanged and solutions found in a consensusoriented manner. You can now hold the result of this not always easy process in your hands!

We hope that this guideline does justice to the topic of Open Source Software in a way that is both generally comprehensible and still meets high professional standards. It is our explicit intention to continuously update the document in the future in order to keep it up to date. We therefore cordially invite all interested parties to participate in the further development of the guide. We are always available for questions and critical comments on the contents.

- Dr Ing Marius Brehler, Fraunhofer Institute for Material Flow and Logistic IML
- Prof Dr Christian Czychowski, Nordemann Czychowski & Partner Rechtsanwältinnen und Rechtsanwälte Partnerschaft mbB
- Sebastian Dworschak, Nordemann Czychowski & Partner Attorneys at Law Partnership 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 Institute for Communication, Information Processing and Ergonomics FKIE & University 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

Acknowledgement

The Bitkom guide to Open Source Software in version 3.0 builds on the previous version. At this point, we would also like to thank the authors of version 2.0 from 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, June 2022

Changelog

All notable changes to the Open Source Guide are documented on this page. Its format has been adopted from ↗ Führe ein Changelog. The version numbers of this guide are assigned according to ↗ Semantic Versioning assigned.

[3.0] – 30.09.2022

Changed

- The entire guide has been fundamentally revised compared to previous versions.

[3.1] – 28.02.2023

Changed

- The chapter »Software Bill of Materials (SBOM)« was added as chapter 9.2
- The chapter »Excursus: The emergence of Open Source Software« has been moved to
- moved to level 9.1.
- A »Changelog« page has been added.
- The index has been updated

1 Introduction | Preface

The original version of this guide dates back to 2016, and six years later it was high time to revise it. Because in the meantime, developments around »open source« have accelerated massively and the realisation has matured on a broad front that digital innovation is not possible in a cost-efficient way without Open Source Software.

This was also clearly shown by the *Bitkom Open Source Monitor*, the study on Open Source Software in Germany conducted for the second time in 2021.[1] Open Source Software is used – across industries – in almost every development project and is a component of almost all software products. Active participation in open source projects and the strategic use of open source methodology is also widespread today. In other words, open source has won across the board.

In order to do justice to this reality, the Bitkom Open Source Working Group is presenting an extensively revised new version of its guide.

This guide provides a broad overview of the widely established practice of open source >use and its legal implications[2]. It highlights experiences with Open Source Software, points to *best practices* for active participation in open source developments and outlines ways to manage open source projects. It also highlights opportunities for the strategic use of Open Source Software and methodologies, right into new business models.

Using Open Source Software successfully ultimately means playing the cooperative open source game of free use, free distribution and free modification. Those who play along will experience that their own benefits far exceed the costs resulting from the licence-related demands. Participants in this »game« will experience that they get more out of the community than what they invest. So everyone can do the math and see their personal positive balance.

In this broad sense, the guide is aimed at

- Decision-makers,
- Project leaders and
- Developers

from business and administration who want to use Open Source Software and open source methodologies and tap into their benefits.

The board of the Open Source Working Group hopes that you will enjoy reading our Bitkom Open Source Software Guide, version 3.0, and that you will gain good insights. The entire board is at your disposal for questions and comments, and we cordially invite you to participate actively in the Open Source Working Group.

1 ↗ https://www.bitkom.org/opensourcemonitor

2 For a supplementary consideration of the legal aspects, please refer to the Quick Guide ↗ »Open Source Software – Legal Basics and Action Guidelines«

2 Info & Basics

2.1 Relevance of Open Source Software

In the digital age, the relevance of software is ubiquitous. In the highly industrialised countries, there are hardly any areas of life, work or leisure in which digital processes, assistants or widgets are not established. What we call »digitalisation« nowadays is largely run and driven by **Open Source Software**: billions of people have Open Source Software in their pockets in the form of a smartphone operating system. The cloud largely runs on Open Source Software[3]. It has even arrived on Mars as part of the Mars helicopter Ingenuity[4].

Studies show that Open Source Software is used directly or indirectly in almost all companies. For example, the Bitkom Open Source Monitor 2021[5] found that 87% of all large German companies consciously use Open Source Software.

87 %

of all large German companies consciously use Open Source Software.

Open Source Software is in no way inferior to proprietary software in terms of **reliability and security**. In many areas, it shows itself to be an **innovation engine** with enormous potential. Alongside the **diversity of application scenarios** is the **abundance of applications**. Open Source Software has gained **real economic significance**.

The reasons for this are simple:

- The software is free to use, there are no licence fees, nor is there any risk that the use of the current version will be restricted by the copyright holder in the future.
- By using standardised open source licences, no individual contract negotiations are necessary.
- Open development models give users more control and guarantee the independence to maintain and further develop the software themselves or through service providers (afterwards).

Even manufacturers who produce internet routers, televisions, car entertainment and other **hardware systems** in large quantities use Linux almost exclusively nowadays. Proprietary systems have been pushed back into specialised niches. Every company can mutually access all innovations in Linux. This is precisely what makes open source remarkable, by upholding cooperation and collaboration as well as the sharing of results: technologies developed in an open process are available to all. Consequently, a company no longer has to build and maintain the complexity in their own system independently. Instead, everyone gives a small part and receives the entirety. This redu-

3 Cf. ↗ https://www.golem.de/news/open-Source-microsoft-hostet- mehr-linux-als-windows-vms-in-azure-1907-142245.html
4 Cf. ↗ https://github.com/readme/nasa-ingenuity-helicopter
5 See ↗ https://www.bitkom.org/opensourcemonitor

ces the company's own development costs, and competition shifts towards strategic specialisations and service offerings.

This shift towards a shared, collaborative delivery of the technology itself and the focus on a unique selling proposition that manifests itself in a service built on that technology is very evident in the cloud: the so-called cloud-native technologies around Kubernetes and other open source projects are prominent technologies that different companies build on when bringing competing offerings to market – although at the same time they successfully develop the underlying technology together.

This goes hand in hand with the fact that **profit** can very well be made with Open Source Software, because it is a misconception that **Open Source Software** cannot be used commercially. The opposite is true: there are a number of established business models that make use of the advantages of Open Source Software. New business models are constantly emering that benefit from the openness of open source and the enormous reach that can be achieved with open source projects. IBM's acquisition of RedHat in 2019[6] has gone down in recent history as one of the largest acquisitions of a software company. RedHat's business model is based exclusively on Open Source Software.

> It is a misconception that Open Source Software cannot be used commercially

However, these business models cannot be based on the generation of revenue via licence fees and licence agreements. Because even in a commercial context, the essence of Open Source Software of course remains intact: constitutively, this includes that customers **cannot be billed for licence costs**. However, a fee may be charged for the work of compiling and installing software in the sense of a service, as well as for support, maintenance or operation, i.e. general services related to the use of Open Source Software or provided with it.

> With Open Source Software, the business model is shifting away from the pure licensing business.

With Open Source Software, the business model shifts away from the pure licensing business to other business models with and around Open Source Software, such as subscription to services (detailed presentation in ↗ Chapter 5).

Nevertheless, the use of Open Source Software is not free, even if the use itself or the right to use it – financially speaking – is indeed free – and always will be. The effort required to use software responsibly and to ensure that, among other things, licence compliance, security and support are adequately regulated, is reflected in corresponding costs. However, standard approaches have emerged to effectively address these challenges.7

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

7 Read more in the chapters on strategy, compliance and open source management.

2.2 Concept and Definition of Open Source Software

In order to use Open Source Software effectively, it is helpful to first understand the concept of »Open Source Software« itself, which also manifests itself in and with a certain set of terms. Its origins lie in the Free Software movement, which was founded in the 1980s by the American Free Software Foundation. It formulates four fundamental freedoms as a **core idea** that every software must fulfil if it wants to be Free and Open Source Software: Whoever has it must also have the right at the same time,

- to (a) **execute** them,
- to (b) **analyze** them,
- to (c) **modify** them to one's own needs; and
- to (d) **distribute**, even in altered form.[8]

A necessary condition for exercising the four core rights is access to the source code. This is reflected in the term »open source«, which has now established itself, especially in the business environment, as the common term to describe software that fulfils these four fundamental freedoms[9].

The Open Source Initiative (OSI)[10] founded in 1998, developed the ten-criteria **Open Source Definition** (OSD) from this[11]. It is now generally accepted to classify which licences are considered open source. In addition to the right to distribute the software and the availability of source code, the criteria of the open source definition ensure in particular the non-discriminatory use of Open Source Software, so that it cannot be restricted to certain use cases, groups of people or technological areas and products.

The **constitutive rights** of Open Source Software of **use**, **inspection**, **modification** and **distribution** (also in modified form) are not per se associated with software: Software is subject to copyright.[12] According to this, all rights to a software are first due to its authors. Only they can determine what others may and may not do with the work they have created. In order for persons other than the authors to be able to use software in a lawful manner, the author must grant them a **right of use**. The term **»licence«** has come into use for such a right of use, which is regularly linked to certain conditions of use and/or restrictions of use. Thus, if an author – often the programmers or their employers – wants to distribute a software as Open Source Software, he must explicit-

8 cf. the Free Software Foundation Europe's definition of free software, ↗ https://fsfe.org/freesoftware/

9 In addition to »Open Source Software« and »free software«, the contracted term »free and Open Source Software (FOSS)« or »free, libre, and Open Source Software (FLOSS)« is sometimes used. Supporters of the Free Software movement stress that the use of the term »Free« is important to emphasise the ideal values behind the freedoms. In practice, however, especially from a licensing perspective, there is very little difference between »Free Software« and »Open Source Software«. We therefore use the common term »open source« throughout this guide.

10 cf. ↗ https://opensource.org/

11 cf. ↗ https://opensource.org/osd

12 cf. e.g. for Germany §§ 69a ff. of the Copyright Act (UrhG)

ly grant the aforementioned freedom rights. He must place his work under a free licence, an open source licence.

The OSI has also gone beyond the definition and implemented a **»License Review Process«** with the aim of an »Approval«.[13] This means that it is not subject to personal interpretation whether an individual case is really Open Source Software or an open source licence. For more than 60 open source licences, the OSI has ensured that they fulfil the criteria of the open source definition and officially lists them.[14] The OSI has also implemented an »Approval Process«.

This has a practical value: anyone who uses software that has been released under an officially listed licence knows, even without looking at the individual licence, that they have all the rights with regard to this software that the OSD makes a prerequisite. The user also knows that he does not have to take into account any restrictions that the OSD excludes. However, this only applies insofar as it is an unmodified licence text released by the OSD. Which conditions are attached to the exercise of the rights under a licence is only revealed by the respective licence itself. In any case, Open Source Software is ultimately only really Open Source Software if it has been released under an official open source licence that fulfils the ten criteria of the OSD and that has been confirmed by the OSI.

> The conditions attached to the exercise of the rights under a licence are disclosed only by the licence itself.

The question of how to describe software that is not Open Source Software is still open. Common terms are **»proprietary«** or **»commercial software«**. However, the term »comercial software« is misleading, as Open Source Software can also be used commercially and does so on a large scale. Moreover, many business models exist in which open source is an essential part of the commercial strategy. In the following, the term »proprietary software« is therefore used to express that the rights holder reserves essential rights and does not provide users with all four freedoms.

Even if the legal assessment of a software does not depend on its designation but on the respective terms of use, a discourse on licence types has become established. The term »Open Source Software« itself generalises – as does every term. Moreover, in practice one occasionally encounters a certain laxity of speech. This can go as far as deliberate misleading, where attempts are made to profit from the positive connotation of the term »open source« even if no Open Source Software is offered at all.

This corresponds to the label of »open washing«. Anyone who wants to participate appropriately in the discourse should only use the term »Open Source Software« if the software in question is published under a licence officially recognised as open source by the OSI. Misuse of the term »Open Source Software« could be considered a violation of competition law.

13 cf. Open Source Initiative: The License Review Process, ↗ https://opensource.org/approval
14 cf. Open Source Initiative: Licenses by Name, ↗ https://opensource.org/licenses/alphabetical

In this context, it is also worth mentioning a type of licence that has appeared more frequently in recent years, which can be called »source available«. Licences of this type, which have sometimes emerged as a modification of a recognised open source licence, offer much the same rights as an open source licence, including, among other things, the availability of the source code, but exclude specific use scenarios.[15] In doing so, they violate the first fundamental freedom and the non-discrimination prohibition of the open source definition. Thus they are not open source licences and must be analysed and considered as proprietary licences on a case-by-case basis.

15 Examples are the ↗ Server Side Public License (SSPL) or the Commons Clause, which try to prevent public cloud providers from offering the software as a service.

2.3 Opportunities and challenges

To discuss the opportunities and challenges of Open Source Software means to judge. Advantages and disadvantages are always subjective in that they are intrinsically advantages or disadvantages »for someone«. Nevertheless, even people who are fundamentally favourably towards the phenomenon of »Open Source Software« because of their own advantages can take a differentiated view of the situation as a whole. The world is rarely »black and white«. And what offers opportunities can still be accompanied by risks elsewhere. With this in mind, the authors, who gladly and openly see themselves as a team of »open source advocates«, nevertheless want to dare to draw a certainly subjective, but also well-rounded conclusion.

2.3.1 Opportunities

Open Source Software offers advantages that are intrinsic to it – it is only through the associated properties that it distinguishes itself as Open Source Software in the first place:

Four basic Open Source Software rights: Open source compliant licences grant the users of the software licensed in this way farreaching rights. The most important are the »4 basic rights«, namely to be allowed to use the software, for any purpose, to be allowed to examine it, to be allowed to distribute it, to be allowed to modify it and to be allowed to pass it on in modified form.[16] The resulting full control over the source code leads to two further important points:

> A good Open Source Software project is therefore developed in public, not behind closed doors.

Transparency: Well-organised Open Source Software projects have, in addition to a source code control system accessible to all, such as Git or Subversion, different code branches for stable releases and ongoing developments, clear releases, publicly available mailing lists, bug tracking systems, wikis and so on. A good Open Source Software project is therefore developed in public, not behind closed doors. However, the fact that this information is available to everyone does not mean that anyone who wants to can have write-access to the source code control system. In many Open Source Software projects, only a few people have direct participation rights. It is true that everyone may contribute to the success of the project by making changes. But often these »preliminary works« are vetted and only included into the project's sources after being reviewed by separate integrators. This ensures that only code that has previously

16 cf. ↗ https://www.gnu.org/philosophy/free-sw.html

gone through a peer review is included in the project's source tree. On the one hand, this ensures quality. On the other hand, the procedure ensures that each code sequence can be assigned to a person. This is particularly important for licensing issues and copyright-relevant actions.

Such transparency increases trust in software, which is a necessary success factor, especially for software that handles sensitive data.[17]

Quality: Open Source Software is often characterised by high quality. The transparent development processes enable this quality through the diversity of the developer community and their interaction through peer reviews of concepts and code. To increase efficiency, open source projects are usually also pioneers in the automation of test, build and release processes. This contributes significantly to the quality of the software.

Increased speed of innovation: Many Open Source Software packages implement new features very quickly and are in themselves an innovative solution. This may be a disadvantage for some software manufacturers insofar as they compete with the Open Source Software packages with their proprietary software. For manufacturers who integrate these innovative open-source software packages as part of their own products, however, the advantage is twofold. On the one hand, innovative features are quickly available in the product and on the other hand, by relieving the burden on their own developers and the project budget, resources are freed up that can be invested in the development of further Unique Selling Points (USPs) of the product.

Reusability: Open Source Software not only increases the speed of innovation, but can itself be an incubator for innovation. Existing Open Source Software solutions can be a building block or basis for further Open Source Software products. Here, too, resources can be saved or freed up and thus put into the actual innovation.

Faster release: Products that contain Open Source Software can be brought to market more quickly. The use of Open Source Software relieves the burden on in-house developers and the project budget. In addition, well-maintained Open Source Software packages are usually also well tested and have a higher test coverage than the selfdeveloped alternatives. Using Open Source Software therefore usually helps to achieve product stability.

Open Source Software not only increases the speed of innovation, but can itself be an incubator for innovation.

17 As an example, the Corona warning app for Germany has been developed as a transparent open-source project to increase acceptance for its use among the population. ↗ https://github.com/corona-warn-app

This is also due to the fact that the application spectrum of Open Source Software is in principle broader. Self-developed solutions are often specifically tailored to a product or to the company's field of activity. If the broader properties of Open Source Software are cleverly exploited, products can be brought to market earlier as a result.

Independence from the manufacturer: Users of Open Source Software packages usually have greater room for manoeuvre. Errors (bugs) can be reported on the mailing lists – often »well« reported errors are quickly fixed by the developer community. However, reporters are not entitled to have the bugs fixed (see risks). Instead, own developers or explicitly commissioned software service providers can fix the bug or contribute the enhancement. The same applies to maintenance. In addition, commissioned service providers can be replaced more easily because the basis of what they do, the software, is freely available.

No licence fees: No licence fees need to be paid or demanded in order to use Open Source Software.

No licence fees need to be paid or demanded in order to use Open Source Software.

Advantageous aspects in the company are:

Easy integration and adaptability: Because the source code of Open Source Software packages is available, they can be easily integrated into the usually heterogeneous IT landscape of a company and/or adapted for specific use in a product. Open Source Software packages are, so to speak, rough diamonds that only need to be polished for use in companies and products.

Many prominent Open Source Software projects pose a reference in terms of solving programming challenges.

Increase of competence of own employees: Your own employees can increase their own programming competence by analysing the source code. Many prominent Open Source Software projects pose a reference with regard to the solution of programming challenges. Consequently, the solutions realised in the projects can be used as templates for solving similar problems.

Cross-company opportunity for standardisation: Much of the software used in products has no differentiating factor, but is needed to ensure the functionality of the product. It is this software that forms the basis for cross-company defacto standards that are developed as Open Source Software in cooperation with other market participants. This approach enables everyone to reduce the effort for these non-competitive parts and still create a stable basis for product development. The freed-up development capacity can be used for differentiating functionalities, which strengthens the market opportunities of the participating companies.

However, by collaborating on the Open Source Software, the influence of the participating companies on the software is also guaranteed, the company keeps its finger on the pulse and avoids the risk of being surprised by changes. This prevents uncalculated efforts to rectify structural discrepancies between the open source base and the company's own proprietary software parts.

2.3.2 Challenges

Security: Open Source Software users are not spared having to deal with a particular discrepancy: Anyone who uses Open Source Software soon encounters the question of how Open Source Software can guarantee security, when all possibilities of misuse and exploitation of programming errors are, according to the concept, made freely and openly available. There is nothing to doubt about this:

With Open Source Software, things are – by the very nature of Open Source Software – really out in the open, more out in the open than with proprietary software, where the source code remains under lock and key. To conclude from this that Open Source Software is less secure than proprietary software and that proprietary software is therefore inherently more secure is a mistake. For the users of proprietary software simply do not know what possibilities of misuse and exploitation of programming errors are contained in it. They certainly do not know whether these possibilities for misuse – intentionally or unintentionally – have not already been passed on to third parties by the manufacturers. To conclude from the fact that something cannot be seen that it is not there is illusory. Users of proprietary software are dependent on assurances from suppliers in this regard.

For users of Open Source Software the situation is quite different. Precisely because all sources are open, they can – if it is important to them – check the security of the software themselves. And even if it is not very important to them, they can still trust – at least in the case of widely used Open Source Software – that others have actually used this possibility of control. This also shows that with Open Source Software – triggered by the community – improvements and corrections are often made available in a much shorter time than with (commercially driven) proprietary software. Thus, on closer inspection, the supposed disadvantage actually turns out to be an advantage.

With the large number of open source modules in use, which can easily number in the thousands, another challenge arises: how can vulnerabilities in one of the many modules be prevented from being overlooked? There are essentially two dangers here: targeted attacks by infiltrating manipulated modules with malicious code, so-called supply chain attacks, and the use of outdated versions with known vulnerabilities. These dangers can be countered with suitable processes and tools for automation.

Open source licences offer extensive freedoms, but link these with some obligations.

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.

A specific challenge for Open Source Software users concerns the possibility that by using Open Source Software in marketed software products, one's own »core know-how« must be disclosed because of the copyleft effect established with the Open Source Software licence (reference to ↗ chapter 7.2). Business-critical unique selling propositions may be unintentionally »generalised« in this way. That this possibility exists is first of all a fact. Nevertheless, on closer inspection, this »danger« does not appear to be too important: in most cases, the value of the unique selling proposition of one's own business in comparison with competitors is small in scope – even if it is subjectively of great importance as an objective »triviality«. Conversely, this means that the larger part of one's own business can very well be realised in and with copyleft-afflicted Open Source Software without offering competitors an unwanted advantage. In addition, it must be taken into account that the copyleft effect only extends to own code in certain usage scenarios and technical architectures. It is therefore important to differentiate precisely.

Support: For the reliable and secure operation of software, it must be ensured that the necessary support exists to be able to correct critical errors and security problems in a reasonable time. This is usually offered by software manufacturers as a commercial service.

Open Source Software offers additional possibilities. On the one hand, other providers can also be commissioned, since the openness of the code means that not only individual manufacturers have the competence and rights to maintain the code and thus offer corresponding support services, but this can also be done by third parties. On the other hand, the users themselves can slip into this role and provide support independently. The community behind the project is often a great help here. However, it must be carefully weighed up which guarantees are necessary in a specific case and who can provide them reliably and within the required time frame.

Open source licences give extensive rights to the published state of the code, but exclude liability and warranty.

Sustainability of open source projects: By publishing Open Source Software, one does not enter into a formal obligation to maintain this software in the long term. Open source licences give extensive rights to the published state of the code, but exclude liability and warranty – at least as far as the national legal framework allows. Furthermore, they do not specify the timing, nature and extent of future development, except that some licences may require future code to be published under the same licence.

This can lead to problems on the part of users who use Open Source Software over a longer period of time and thus depend on open source projects also being developed sustainably. Two aspects in particular need to be considered here, the **governance** of projects[18], as well as their **funding**. As a responsible user of Open Source Software, one must take this into account both when selecting projects and, when using projects, find ways to contribute to the sustainability of the projects and thus maintain the basis for reliable maintenance of the projects.

2.3.3 Pitfalls

Finally, there are challenges that are sometimes presented as specific »problems« of Open Source Software, when in fact they also affect proprietary software in the same or similar sense. We list some of these aspects:

Licensing entanglements: Software packages today are complex. They contain components that are not necessarily distributed under the main licence. At a time when more and more commercial products are also distributed in combination with open source components, this challenge arises for open source and proprietary software alike.

Legal side effects: Open Source Software can – even unintentionally – infringe third-party rights. Programmers can use patents quite unintentionally without acquiring patent licences. However, again, this applies equally to open source and proprietary closed source software: both types of software can infringe the (patent) rights of others, so that the users of the software are challenged for merely using it. In the case of software that is commercially purchased from a manufacturer, however, one believes to have a claim opponent for such indirect damages in and with the supplier's warranty. Whether the financial strength of the supplier is sufficient to actually cushion on this damage, including the associated legal expenses, requires separate consideration.

> Any violation of the transfer obligations entails the risk of permanent withdrawal of the rights of use.

Revocation of rights of use: Every violation of the transfer obligations entails the risk of permanent revocation of the rights of use. This can happen with proprietary software as well as with Open Source Software.

18 See also the section on open source foundations that contribute to the sustainability of projects by establishing open governance.

Limited liability: Open Source Software licences always contain a socalled disclaimer of liability. Due to legal peculiarities, this exclusion is reduced in Germany – very roughly speaking – to the liability promises that accompany a gift. Commercially distributed software is subject to stronger obligations. Nevertheless, their liability options are also reduced in terms of commercial strength, while, conversely, commercial distributors of Open Source Software enhance precisely the original limited Open Source Software liability as one of their business models.

Unkown follow-up costs: Unestimated and unplanned costs, such as upgrade or security-related reintegration work, that arise from the use of Open Source Software can call the business case of a product into question. But it is certainly also true for proprietary software that unestimated and unplanned costs corrupt the business case of a product. What is certainly not true (for both), however, is that users are necessarily faced with costs that cannot be estimated or planned for. Here as well as there, a proper analysis of the consequences is required, both when using proprietary software and when using Open Source Software.

Know-how deficits: The ease of using open source packages sometimes tempts people to underestimate the effort required to integrate them into their own product and to think that they can be reproduced at any time. However, this work requires expertise. What is, so to speak, »automatically« included and remunerated in the service in the case of commercial use, must also be organised with regard to packages obtained directly from the open source project. Doing without this is tantamount to doing without manufacturer support.

The ease of use of open source packages sometimes tempts one to underestimate the effort required to integrate them into one's own product.

Outdated software: Open Source Software can become outdated, be it in development, be it in documentation, be it in distribution. Suddenly, users are faced with »dead« packages that are no longer maintained. However, there is also a similar case on the proprietary side. Companies can also become »obsolete«. They can close down, change their focus or be bought up. And just as suddenly, the customers of proprietary software are also faced with »dead« software. On the open source side, however – because of the four freedoms – there is at least the possibility of a »revival« in principle.

Negative press: In the case of Open Source Software licence violations, there is certainly the possibility that the software scene will react violently in terms of communication and thus damage personal reputations. Conversely, the suppliers of proprietary software also have fierce means at their disposal in the event of a violation of their contractual terms, except that these are likely to be primarily legal.

Troubleshooting: There is no entitlement to troubleshooting vis-à-vis the community.

In many cases, you will still receive excellent support from the community and, in addition, support with guaranteed reliability is often offered in return for payment.

These few examples show that there are also challenges with Open Source Software. However, they also show that corresponding challenges must also be managed on the side of proprietary software. In terms of »challenges«, open source is not quite as special as is sometimes claimed; in terms of »opportunities«, however, it is very special.

3 Benefits of Open Source Software

3.1 Strategy examples for the use of Open Source Software

The use of Open Source Software in one's own IT should be well thought out. In the course of this guide, we highlight various benefits and pitfalls of open source and also devote an entire chapter to an overarching open source strategy.

However, if you only want to use Open Source Software and are not yet thinking about the »big picture«, this section provides food for thought and examples of how open source can find its way into your own company.

3.1.1 General rejection

The use of Open Source Software components is not permitted in the company.

Open Advantages	Open Disadvantages
Isolation from unknown distribution, cooperation and licensing models.	No use of freely available goods, implementation and control costs of the strategy are not necessarily lower than the costs of a strategy »pro« Open Source Software.
	Open Source Software is a component of most proprietary software and hardware products. As a user, doing without Open Source Software is therefore very costly and likely impossible.

3.1.2 No open source in own products

Bringing Open Source Software into the supply chain (embedded in own products) is not allowed.

Open advantages	Open disadvantages
Strict avoidance of involuntary publication of unique selling propositions.	Use of all freely available goods in self-development excluded.
Use of open source by suppliers possible, whereby risks, support and maintenance expenses can be shifted to them.	

3.1.3 Only selected open source licences in own products

Bringing Open Source Software into the supply chain is possible, provided that the conditions of the respective licences and maintenance aspects are accepted by the company **as a lump sum**. Typical: No components with strict copyleft.

Open advantages	Open disadvantages
More fine-grained use of open source components possible.	Different product and risk groups require different approval classes. A simple allow/deny list often reaches its limits here.
	Monitoring of the components necessary.

3.1.4 Selected Open Source Licences in Selected Products

Bringing Open Source Software into the supply chain is possible, provided that the conditions of the respective licences and maintenance aspects are accepted by the company on a **case-by-case or use-case basis**. Typical: No copyleft in embedded devices and only if the copy-left effect does not come into play due to various factors.

Open advantages	Open disadvantages
Most correct use of open source components possible.	Case-by-case testing, staff training and fine-grained models necessary.
	Wrong decisions possible and expenses often high.
	Detailed monitoring of the components necessary.

3.1.5 General acceptance of open source in own products

The use of Open Source Software is permitted.

Open advantages	Open disadvantages
Full access to the entire pool of open source software.	Pure use is not a sustainable strategy in most cases.
Full cooperation with the community is possible.	Cooperation with the community remains unsystematic.
	There is a danger of falling into the »maintenance trap«: Occasionally, Open Source Software is still modified or extended before it is integrated into one's own products. If these modifications and extensions are not then returned to the corresponding Open Source Software projects, they have to be integrated again into the updated software version after an upgrade of the Open Source Software base. In the end, this is »unproductive« work.

3.1.6 Openly strategic use of open source in own products

Open Source Software is of strategic and competitive importance. The company will become a valued and active part of the open source ecosystem and its products and services will benefit from the technical and economic advantages of Open Source Software.

Open Advantages	Open Disadvantages
Full access to the entire pool of open source software.	A longer time frame will have to be estimated for the implementation of the strategy, which means that it is a »long term investment«. All those involved must be aware that the investment will only really pay off in the longer term. In this context, it is worth recalling the definition of strategy as »achieving a medium-term or long-term corporate goal«. Accordingly, the long-term nature of a strategy is not a disadvantage, but an inherent characteristic; unfortunately, this is often forgotten.
Full cooperation with the community is systematically designed to meet the company's goals.	The strategy is only promising if a rethinking process takes place in the minds of all those involved and responsible and there is a willingness to go through a certain learning curve. Furthermore, one must be aware that parts of the company may become more transparent as a result. This is not a disadvantage in itself, you just have to know it in advance and be able to deal with it.

3.2 Standardisation and customer protection

3.2.1 Certifications

The term »open source« itself is also successful as a label and often serves as a marketing tool. However, not everything declared as »open source« deserves this title – at least not according to OSD or the FSF's definition of free software. The certification of licences by the OSI (cf. ↗ Section 2.2) fulfils an important standardisation function: The use of an appropriate licence guarantees providers and users of Open Source Software reliability with regard to compliance with the respective OSI specifications.

Occasionally, however, Open Source Software appears that was created on the basis of an established licence from the BSD licence family, but with deviations so that the result may no longer comply with the OSD. The software in question may still be open source but subject to other restrictions, such as a ban on further commercial use or the exclusion of certain user groups (such as defence companies, intelligence services). Providers should therefore opt for a licence certified by the OSI – especially since the official OSI list contains a corresponding licence for almost every application. Users should make sure that the software is licensed under one of the listed licences. This minimises uncertainties and potential risks for both sides, but does not exempt them from checking the concrete use in each individual case. This applies especially to unusual licence obligations of licences that do not comply with the OSD.

By now, the press and the public are sensitised to abuses of the label »Open Source Software«. Those who use the labels »open source« or »free software« should be aware that the internet and social media quickly expose cases where this is only a façade. The potential damage to the image can be higher than the benefits that should be achieved through the declaration.

Certifications play a not insignificant role in the field of Open Source Software and contribute to standardisation. On the one hand, certifications for (legal) persons can be considered, and on the other hand, those for the open source licences used. From the analysis of possible open source business models (cf. ↗ Chapter 5), it is clear that companies need different criteria to distinguish their offerings based on Open Source Software than is the case with proprietary approaches based on license fees.

Certifications play a not insignificant role in the field of Open Source Software and contribute to standardisation.

Proven expertise is one such criterion that can create advantages over competitors. The qualification of the employees becomes clear through the examinations taken at the Linux Professional Institute (LPI)19. In associated courses (some of which are self-study), those interested acquire knowledge at various levels of experience and, if successful, receive a certificate. With the LPI,[19] the learning materials and the corresponding exams, there is thus a uniform, standardised knowledge base. This also facilitates the writing of job offers and the assessment of an applicant's level of knowledge. For companies, for example, ISO-523020[20] stands as a seal of Open Source Software compliance.

3.2.2 Support services for Open Source Software

Open Source Software generally differs from proprietary software in terms of support services. The main reasons for this are, on the one hand, the often decentralised development of the software without a specific contact person for the users and, on the other hand, the fact that the software is usually made available free of charge for use or further development.

> To close the gap of missing warranty claims, it may be advisable to make use of paid service and support services.

While the warranty for software distributed in return for payment is governed by the statutory provisions of the law on sales or contracts for work and services, the warranty for software provided free of charge, such as Open Source Software, is limited. As a rule, it is governed by the legal provisions of the gift. As a result, the manufacturer is only liable for defects if it can be accused of intent or gross negligence or of fraudulent concealment of a material or legal defect. A defect of title exists in particular if the use of the software is opposed by the rights of third parties. The advantage of no licence costs is therefore offset by limited warranty claims.

In order to close the gap of missing warranty claims that arises with free software, it may be advisable to make use of paid service and support services. The palette of such offers is diverse – it ranges from troubleshooting to closing security gaps to further development and individual customisation of the software.[21]

Depending on the content of the contractual agreement, different types of contracts can be considered with regard to service and support services: If the services are performance-related according to the agreement of the parties, it will generally be a contract for work and services. In this case, a concrete performance result or an activity success is owed, for example, the elimination of faults or errors and the maintenance of the functionality of the Open Source Software. If, on the other hand, it is merely a atter of the conscientious implementation of the measures, for example simple

19 cf. ↗ https://www.lpi.org
20 ↗ https://de.wikipedia.org/wiki/ISO/IEC_5230
21 cf. ↗ Chapter 5 on business models

consulting services in the use of Open Source Software, a service contract comes into consideration. Since in this case no tangible result is owed, care should be taken to specify the quality of the service and support to be provided.

A clear classification as a contract for work and labour or a contract for services is not always easy and ultimately depends on the design of the contractual relationship in detail. The parties involved should make sure that the agreements are as clear as possible in order to create a secure basis for the joint cooperation and to avoid disagreements in case of deficiencies in performance. If a contract for work and services is desired, acceptance in particular should be expressly regulated.

A clear classification as a contract for work and labour or a contract for services is not always easy and ultimately depends on the design of the contractual relationship in detail.

It should not go unmentioned that with Open Source Software, users always have the option of providing support themselves, since open source licences grant all the necessary rights, in particular the right to modify the code and to pass on modified code. If the necessary competence and resources are available, this can be an alternative to purchased support services.

Support for Open Source Software is often in demand enough to make it a business model in its own right. One criterion for evaluating support service providers is contributions to the open source projects covered by the support. However, it should be noted that this is not the appropriate criterion for every form of service. For providing security updates it is a good indicator, for consulting services not necessarily. In any case, funding a support service provider who actively maintains, services, monetarily supports or contributes features to the software also strengthens the project.

3.2.3 Long Term Support

»Long Term Support« (LTS) offers users a special kind of security: the product versions are provided with important bug fixes and security upgrades for a longer period of time. The reasons why this can be a special challenge are as follows:

Open Source Software is continuously developed by the community in two ways. On the one hand, it involves the provision of new, functionally enhanced versions of the software. On the other hand, it is about providing bug fixes and security upgrades of the functionally non-enhanced version.

Users who do not always want to switch to the latest functional versions immediately therefore have an interest in the version they use being provided with bug fixes and security upgrades over a longer period of time, without this being accompanied by a functional upgrade. Distributors of Open Source Software have developed the concept of »long-term support versions« for this purpose. Here, the distributor guarantees for a certain collection of Open Source Software that it will deliver the desired bug fixes and security upgrades over a longer period than usual.

In other words, LTS versions focus on software quality in order to minimise risk, while new features take a back seat.

Stable LTS versions are usually characterised by a feature freeze. The software status reached is branched off from normal development at a certain point and functionally frozen. While new functions flow into the main development branch of the source code and are already delivered with newer versions, only bugs and weaknesses are fixed for the branched-off status. The corresponding patches can be made available individually or in new releases (maintenance or service packs, minor releases, etc.). Therefore, updates are less frequent and consist – if at all – of functions that have already been extensively tested. This is to minimise the risk of new bugs creeping in or previous functions being unintentionally compromised or even taken out of service. A typical cycle for such an LTS release is two years – but can also be shorter or longer or relative to other releases.

The feature freeze variant should be considered by Open Source Software providers in their own LTS strategy, both in terms of the duration for which old versions are still supported and in terms of the release of special LTS versions. Meanwhile, another business model is to offer commercial LTS beyond the period guaranteed by the community.

Users of Open Source Software should inform themselves about the respective rules of the LTS. Almost all projects have their own policy, such as the Eclipse Foundation or the Ubuntu distribution created by the Canonical company. For long-term IT and product planning, the information to be gained from this is of great value.[22]

22 A special case is represented by manufacturers of customer electronics who use Linux on their systems: Here, a cross-company Long Term Support Initiative (LTSI) was launched in the form of an industry-wide project. This project defines Long Term Support for an industry branch of the Linux kernel, which serves as a reliable basis for customer electronics. (cf. for example: ↗ https://ltsi. linuxfoundation.org/what-is-ltsi)

3.2.4 Protection against third-party claims

The use of Open Source Software can lead to the infringement of third party rights.

The use of Open Source Software can lead to the infringement of third party rights. These include in particular third-party copyrights and patent rights, but also trademark rights. Since Open Source Software is not usually examined and analysed in detail with regard to such legal positions before it is used in practice, there is a risk, which should not be underestimated, of infringements of rights and thus of expensive warnings, injunctions and claims for damages as well as cost-intensive patent disputes initiated by the respective rights holders. Ultimately, this can affect everyone – those who distribute Open Source Software and those who use Open Source Software.

In ↗ section 7.3.2, OpenChain and ISO 5230 are discussed. This is an industry and ISO standard for reducing open source compliance risks in the enterprise and supply chain.

In the case of software provided free of charge, the licensor is only liable to a limited extent for defects in title. If neither intentional nor fraudulent action was taken – which is often the case – the users or distributors are left with the damage; in this case they alone bear the risk of defects.

In this case, some distributors offer special insurance cover for a fee. Such products, marketed under the name »Indemnification Program« or »Assurance Program«, for example, grant Open Source Software users additional claims and protect them financially against the risks mentioned. Such insurance policies usually cover damages arising from patent and copyright infringements to a certain extent. In these constellations, the risk of infringement of third-party rights is also reduced by the fact that typically more effort is put into checking for such possible third-party rights by the distributors.

In individual cases, however, it should be checked whether these insurance products also cover the concrete intended use of the software. Care should be taken to ensure that claims are not limited to the case of personal use if the software is intended to be distributed individually or in a package with other components. Also, the insurance may only cover a core of the distribution, but not all programme parts contained in the distribution.

3.3 Quality Criteria to Use Open Source Software

Anyone wishing to use Open Source Software should not disregard operational aspects and quality criteria of any software in open source. This is especially true since Open Source Software comes without SLAs or contractual assurances or liability of any kind when used directly.

Particularly in the case of frequent or critical use of frameworks, libraries, infrastructure and services, it quickly becomes relevant whether the software is linked to a company, whereby service and support can be purchased in case of doubt, whether a lively and diverse community exists and helps out, which may express financial needs, or whether no one or only a few see themselves as responsible for the software.

There is no conclusive list of criteria, because what is relevant to a software always depends on its use. However, some aspects are frequently encountered and are summarised below.

- **Governance:** How are decisions made in the project and who has what influence? Larger projects have many stakeholders who all have some form of influence on the project. Who leads the project? What intellectual property arrangements are in place? Who pays for infrastructure and/or maintenance? How this is implemented usually has to be examined on a case-by-case basis. However, it can be uniformly said that the conscious support of the project by third parties, for example by an established open source foundation, often already sufficiently addresses various legal and organisational aspects.

- **Maintainability and support:** Open Source Software can be used free of charge – but it does not have to be. And it is often not directly apparent whether commercial support, as described in ↗ Chapter 3.2.2, is available for a software. Depending on the intended use, it should be determined whether and what kind of support is possible.

- Financial and personnel sustainability: Many open source projects complain about a lack of funding or changing interests of the developers. Often the so-called bus factor is also much lower than one might assume from the reputation of some projects. Those who rely on open-source software should ask themselves whether funding this ecosystem offers added value. It is often possible to combine the pleasant with the useful here, provided that contributors to the software offer paid service and support.

> Those who rely on Open Source Software should ask themselves whether financing this ecosystem offers added value.

- **IT-Security:** Open source is no guarantee of security. It is clear that the reuse of, for example, specialised cryptographic libraries and the possibility of external reviews can enable quality, but this is by no means an inevitable consequence. Those who use open- source software must not forget that the creators may not have experience in dealing with security-relevant aspects or sufficient resources for appropriate optimisations. Software with sufficient documentation, high scores in static and dynamic code analyses and transparently identified mechanisms for quality assurance can often be developed and maintained much more easily and suggests greater experience on the part of the developers. A justifiable or unmanageable bug/issue backlog also quickly provides further information about stability.

- **Contributability:** Open source does not guarantee that you can contribute, and even though most open source projects gratefully accept many contributions, there are also other interests. For example, it is uneconomical for »open core« solutions if the open source solution becomes too good or too complete. Language barriers, different quality requirements, CLAs and long-term ideas about project development can also quickly become a problem. Sometimes even own interests stand in the way of a CONTRIBUTION, for example if the CONTRIBUTION would contradict the image or strategy. Projects often indicate which guidelines apply to the contribution by means of a CONTRIBUTING file.

Various projects approach this problem from different directions. The ↗ Best Practices Program of Core Infrastructure Initiative or the open source tool ↗ fosstars can be mentioned as exemplary criteria catalogues. The approaches shown represent an incomplete list of possibilities for evaluating Open Source Software. It is strongly recommended to use several approaches in combination to form a more comprehensive picture in a selection process.

Important: Which Open Source Software is really important is rarely directly obvious. Often different departments use different and specialised components, but unknowingly communicate via the same cryptographic libraries, are based on the same web frameworks or use the same database. Which component is thus important or even irreplaceable from this point of view often only becomes clear when a department, such as an OSPO, maintains statistics on this.

3.4 Licence Management and Compliance

Regardless of which of the above variants is chosen: If already existing or external Open Source Software forms part of the delivery or publication, the provider must ensure that the licences involved are complied with. The basis for this is licence management, the essential features of which are explained below.

Licence management includes the following aspects:

1. recording of the licences used
2. carrying out licence interpretations
3. management of licence interpretations
4. definition of the implementation possibilities of licence interpretations
5. verification and recording of the implementation for delivery.

More on compliance from a formal point of view can be found in ↗ Chapter 7. In the following, some practical aspects will be examined in more detail.

3.4.1 Recording of licences used

The systematic recording of the external software used and its licences forms the basis of all further licence management activities and measures.

Neither the systematic recording of software from third-party providers nor licence management are specific to Open Source Software, but a general task that every company has to carry out as part of its compliance activities.

All Open Source Software components installed on computers in the company or incorporated in products of the company should be recorded in a central register.

The result would have to be continuously updated and maintained so that changes in the conception are also systematically taken into account. The employees involved with Open Source Software would have to enter the relevant information, including information on the purpose and use of the Open Source Software in the company, always before starting work with the Open Source Software.

A corresponding in-house database should contain the following information:

- Name and version of the Open Source Software,
- Author or source of the Open Source Software,
- Licence type and version of the licence of the Open Source Software,
- Start of use,
- Type of use:
  - Programme or library,
  - Embedded component or stand-alone unit,
  - in modified or unmodified form,
  - with disclosure to third parties or without disclosure,
  - in the form of binary files or as source code,
- Name of the employee involved in the use of the Open Source Software,
- Target project for the use of the Open Source Software,
- planned internal use of the Open Source Software with or without modifications,
- planned external use (copy, distribution) of the Open Source Software with or without modifications,
- Approval of Open Source Software use by the relevant decisionmakers.

In the case of proprietary software, the licence conditions are »negotiated« between the licensor and the licensee, at least in principle. In the case of Open Source Software, the licences and their conditions are generally known in advance and – de facto for the most part – non-negotiable. Thus it is practically a matter for the licensee, i.e. the user of Open Source Software, to record the licences that are relevant for a complete (open source) software package. This means that in case of emergency – i.e. where one cannot rely on preliminary work by the community – the entire package must be searched for licence references.

Due to the complexity and size of today's software projects, a manual search is practically not feasible and also not reasonable. A number of software tools are available that support the identification of the licences used in a software package. Two of these software tools deserve special mention in this context because they are themselves open source projects. These are Open Source Review Toolkit[23] and FOSSology[24].

> Most open source licences require that users of the software are provided with he licence texts with the software.

However, simply identifying the licences involved and the origin of the respective open source packages is not sufficient in the event that the open source package is to be redistributed either in binary form or in source code, for example as part of a product or solution. Most open source licences require that users of the software be provided with the licence texts along with the software. In addition, it is at least good tradition to also name the authors of the software. The way in which this is to be done and in what format is often not defined and is only described in more detail in some licences. The only condition is that this information must be human-readable and deposited in an easy-to-find location.

23 cf. Open Source Review Toolkit (ORT), ↗ https://oss-review- toolkit.org/
24 cf. FOSSology: Advancing open source analysis, ↗ https://www.fossology.org

The lack of a standardised format for providing information to users of the software has so far led to many companies implementing different solutions that are incompatible with each other. This is a growing problem, especially for suppliers, who may have to provide the same information in many different forms and formats. The problem has been addressed for some time by the Linux Foundation within the framework of the »Open Compliance Program«[25]. One element of the programme is the definition of the SPDX exchange format[26], which allows the listing of the components used, their files, as well as the licences involved and the corresponding authors. This standard is available in version 2.2.1 as ISO Standard 5962:2021. This procedure of a structured recording of used open source components is known as Software Bill of Materials (SBOM). It is increasingly becoming a central tool in the open source compliance process.

Ultimately, the open source input control at the companies may be limited to the evaluation of such documents – provided they were created by the suppliers with appropriate care or automated with the help of suitable tools. Nevertheless, it is important to note that it is the duty of the using company to actually fulfil all conditions of an open source licence. It is not possible to shift this obligation to the suppliers.

This structured and ideally standardised recording of open source components in use is also known as the Software Bill of Materials (SBOM).

3.4.2 Resilience of licensing information of different open source ecosystems

When talking about recording licensing information, it is assumed that open source projects themselves report sufficient licensing information. However, the reality is often that in different ecosystems, especially those that are fast-moving or complex, or that want to make it very easy for open source entrants, there are often insufficient rules for providing open source licence information.

In some ecosystems, for example, it can be observed that code is often published without licences or that licence information is incorrectly or incompletely reported.

While open source operating systems often make rather high demands on compliance and also push it, other distributors already show significantly less enforcement of licence types and information. In the end, there are distributors of complete compositions such as container images, which are usually non-transparent with regard to licences and content.

25 cf. Linux Foundation: Open Compliance Program, ↗ https://compliance.linuxfoundation.org/
26 cf. ↗ https://spdx.dev – the standard is also applied in this guideline

Anyone using open source needs to make sure which ecosystem he or she is dealing with, and which subcontracting he or she has to expect, or which risks are to be expected in the data situation. This is often where the desire for compliance and the reality of the data situation collide. For example, theoretical requirements may be trivially feasible in a few ecosystems and at the same time lead to disproportionate expenditure in others.

Nevertheless, there are various possible solutions. Those who use open source for exploratory purposes and in the low-risk area can often find an economically appropriate middle ground between complete and forensic compliance and accepted risk. In addition, it is often a good idea to combine open source and security, because often the most opaque solutions in terms of licensing law are equally opaque for various security measures. An example of this could be container images, which are only released for use after explicit testing or from a controlled internal build.

3.4.3 Container compliance

Containers are a method of encapsulating software, both at runtime and for distribution. Sometimes containers are described as a lightweight form of virtualisation, but in fact significantly different technology is used, which brings specific compliance challenges.

The central advantage of containers is their ease of handling. With just a few commands, container images can be downloaded, further container images derived, a container instantiated or container images published. Systems are built from a multitude of containers, integrated and orchestrated via specialised tools.

> The central advantage of containers is their ease of handling.

The former hype has grown into a serious technology that will be an important foundation for future innovation. Workflows, testing, test operation, rollout and productive operation are considerably simplified, and the massive open source content enables a high level of transparency and adaptability. In addition, containers have reduced resource requirements, especially compared to virtual machine images.

The offer to use container technology here covers the entire range of possible uses – from purely internal use to worldwide deliveries of solutions with various supporting support measures, to »self« updating complete solutions already in implementation.

But the speed and elegance of a new technology has often been a deceptive protection against oversights and the violations that come with them.

The simplicity of container technology makes it possible to pull together software from a wide variety of sources, even in an unstructured way. Each Linux distribution handles licence and copyright information strikingly differently and at very different levels of quality or detail. Patch policies are fundamentally different, following different philosophies and necessities. In addition, software can also easily be integrated directly into containers without using consistently built packages of a Linux distribution.

This creates a great potential for omissions, both in their own development and in the development of the suppliers, i.e. the entire supply chain. Because while a container hides the complexity of a software stack in its external appearance, the legal, regulatory and contractual obligations still apply to all components used and their interaction. Unnoticed, this new technology could undermine existing corporate policies. A possibility that should not be underestimated.

The simplicity of container technology makes it possible to pull together software from a wide variety of sources, even in an unstructured way.

From the point of view of licence compliance managers or IT law specialists, many of the current container results can be compared to Pandora's box:

- Container content is often obtained from a variety of different sources where trustworthiness of sources and content cannot be readily assured.
- Often licence references are blurred, incomplete or have even been completely removed (keyword: pico-container).
- The question of plausibility, compatibility and fulfilment of licensing obligations remains unanswered already at the source of a basic container image.
- Without further information, tools and effort, the company can hardly come to a conclusion about how secure the software in the container is and which known security vulnerabilities may be relevant.
- The technology goes beyond the scope of a container. This means that the licence of each container image and the content it contains must be kept track of.

Without the appropriate support tools and a dialogue between the disciplines in the company, a realistic assessment is hardly possible. Moreover, every company should urgently come to its own legal understanding and its own assessment and positioning vis-à-vis these new technologies as well.

In this light, a one-sided ad-hoc assessment, immediate use or disclosure appear unprofessional and negligent.

In principle, the situation is not hopeless. The basic procedures for open source compliance can also be applied to containers. However, the peculiarities of container technology pose special challenges, for example:

- The construction of containers from different so-called layers requires that, in principle, licence compliance must be established for each layer.
- There is a lack of standardised procedures for the distribution of associated source code, in contrast to, for example, Linux distributions, which offer mature procedures for this purpose.
- Signature procedure of containers to ensure the integrity of the contents is not very mature yet

A unified position of the developing industry in Germany and internationally is not yet discernible in this context. There are a number of tools available, but they cannot yet cover all compliance issues. In order to remain future-proof, capable of action, competitive and guilt-free, container compliance must be viewed with a high degree of attention.

In order to remain future-proof, capable of action, competitive and guilt-free, container compliance must be viewed with a high degree of attention.

If a company plans to use container-based solutions in a productive operation or to pass them on to further recipients in a supply chain, the implementation of the following recommendations for action is desirable:

- External container images are only obtained from selected sources.
- External container images are to be verified according to own guidelines.
- Derivations of own container images can be reproduced via defined processes.
- Modifications to third-party components are restricted by own guidelines and are checked by reviews or tools when generating the derivations.
- Active inventory of software components – especially third-party components – to check against own policies.

The contents of the listed own guidelines are left open here. Ultimately, the objectives of these guidelines are a matter for the company or result from the regulatory, legal and contractual framework conditions from the perspective and risk appetite of the company's management.

With the current heterogeneous content, existing standards and quality arguments, we are still a long way from a microservice architecture based on a serverless infrastructure that bears the predicate compliance-by-design and does justice to society's growing understanding of transparency.

3.4.4 Implementation and Management of Licence Interpretations

The systematic recording of the licences to be applied is followed by the interpretation of the rights granted and obligations imposed in the licences as well as the administration of the interpretation. The aim of interpreting the licences is to avoid copyright infringements as well as to get clarity on whether the licences are compatible with one's own intentions. Legal expertise is essential in this activity. The interpretation of licences, as well as their recording, must be carried out systematically and stored in a reusable way to avoid accidental errors. The technology used to manage the interpretations can be chosen freely. However, a database-oriented solution makes sense – especially if the process of recording and deriving the licence conditions to be fulfilled is to be automated as far as possible for reasons of efficiency.

Existing licence interpretations are to be maintained and, if necessary, adapted to newly emerging aspects of case law.

3.4.5 Possibilities for implementing licence interpretations

The interpretation of the licence conditions is followed by the definition of the implementation of the interpretations. The implementation is usually specific to a concrete delivery scenario:

> It makes sense to define »best practices« in the concrete implementation and to follow these as far as possible in the implementation of the licence interpretations.

In this step, the specifics of the individual products and solutions in which the recorded open source packages are included in whole or in part, with or without modifications, must be taken into account. For example, in the case of an existing GUI (Graphical User Interface), the obligation to include the licence text of the software can simply be implemented by a button »show licence information« in the main menu. If there is no GUI, the licence texts can, for example, be provided on the product CD or on a CD enclosed with the product as text files. It makes sense to define »best practices« for the concrete implementation and to follow these as far as possible when implementing the licence interpretations. This minimises the effort through a kind of standardisation and the risk of incorrectly implementing a licence interpretation in a concrete product. In addition, a company-wide, uniform concept is perceived as a »brand« over time. As an example on the last point: All products of company A are accompanied by a CD called »Open Source Software«, which contains all the details and the corresponding source code of the Open Source Software packages supplied.

It makes sense to define »best practices« in the concrete implementation and to follow these as far as possible in the implementation of the licence interpretations.

An efficient method to ensure compliance with the licence conditions for concrete products and solutions is to treat the licence conditions as requirements that the individual deliveries must fulfil. For example, the requirement of a licence to include the licence text with each software delivery is entered as a »mandatory requirement« of the product in the requirements engineering tool used. The concrete type of implementation for the respective product is documented and tracked in this tool. The highest degree of efficiency and completeness is achieved when the licence conditions (i.e. the result of the activity »performing licence interpretations«) are stored in a database together with the defined best practices, as recommended above. These are automatically read out of the database according to the Open Source Software contained in the respective software delivery and entered into the requirements engineering tool. This type of automation ensures the completeness of the »licence compliance requirements« per product or solution and their concrete implementation is also documented in a traceable and verifiable manner. This in turn significantly simplifies the last activity »verification of implementation options«.

3.4.6 Verification and recording of conversion to delivery

For quantitative reasons, verifying and recording the implementation of the licence conditions to be fulfilled is almost only possible with tool support.

Even in small DSL routers or DVD players, entire Linux distributions with dozens of open source packages are sometimes used. In some language ecosystems, the number of modules used can be several thousand. Due to this sheer quantity alone, it is no longer possible to check whether all licence conditions are being complied with without software support. As described above, all licence management activities should be automated as far as possible. Handling lists and tables in which results, interpretations and conversions have to be entered and maintained manually will inevitably lead to errors, even with small deliveries.

Licence management is an activity that accompanies the deployment of software, no matter what type of software it is – be it Open Source Software or proprietary software. It is important that licence management is done for each package and each release of the packages that is deployed. This is because the type and quantity of licences applied in an Open Source Software package may be different in each release from its predecessor. Attention must also be paid to changing licence versions, as this can entail new licence obligations as well as new rights.

> Licence management is an activity that accompanies the use of software, no matter what type of software it is – be it Open Source Software or proprietary software.

4 Creating Open Source Software

No obligation to contribute to Open Source Software arises from open source licences and, depending on the licence, it is not even necessary to publish changes. Nevertheless, intensive use of Open Source Software will almost inevitably lead to the issue of contribution, as there are a number of very good economic and technical reasons to actively participate in open source:

- Interaction with the open source community and direct contact with open source developers is an excellent opportunity for feedback and learning. This leads to higher software quality and improved ability to provide support for Open Source Software in use.
- In the medium term, a contribution of one's own changes reduces maintenance costs, since after the contribution the change becomes part of the Open Source Software and is thus automatically included in subsequent versions. Without a contribution, the change must be painstakingly updated and, if necessary, adapted with each update.
- Influence on and participation in open source projects can usually only be achieved through active collaboration and contribution of code.
- The sustainability of Open Source Software in use depends crucially on sufficient user participation in its maintenance and further development. The resilience of strategically used software can be improved through participation.
- In many cases, there is also a moral obligation not only to take, but also to give. Organisations that fulfil this responsibility and not only use Open Source Software, but also contribute to it, feel this in the form of an enhanced reputation, among other things as employers for the talents they are courting.
- Strategic engagement in and with open source projects can establish standards and shape markets.
- Open source models can be an effective way to improve collaboration and contact with customers, partners and suppliers.
- Since open source licences guarantee non-discriminatory access to software, this can serve as a basis for consortial software development without hindering or unilaterally influencing the market and thus creating antitrust or other regulatory problems
- For an increasing number of software developers, the opportunity to participate in open source projects can be a strong argument determining the attractiveness of an employer.

The sustainability of Open Source Software in use depends crucially on sufficient users also participating in its maintenance and further development.

A higher commitment to contributions and even the opening of new open source projects arises when these contributions are to support existing or new business models. For more information, see ↗ Chapter 5, where different archetypes of business models are presented.

The topic of open source contributions thus forms a broad field in which there are several aspects to consider. In the following, we will look at the topic of open source project governance, i.e. the management processes around an open source project, as well as other basics, such as project types, the different ways to contribute to an open source project, and typical tooling around communication and development in open source projects.

4.1 Open Source Governance

As a rule, open source projects are not unrestricted source code repositories in which anyone can change software at will. In principle, an open source licence guarantees that the software can be used freely and adapted to one's own particularities. However, it does not define any right to incorporate one's own changes into the original project. For the further development of open source projects, therefore, quite strict review and decision-making processes are often used, by means of which the quality of the software is permanently maintained. These processes, as well as the general decision-making and control mechanisms of an open source project, are summarised under the term »open source governance«.

4.1.1 Contribution pyramid

The community of an open source project typically represents a pyramid. At the base of this pyramid are the users of the software, especially those who actively participate in the community, for example in the form of bug reports and feature requests or through contributions to mailing lists. In the pyramid directly above are contributors. These are members of the community who propose their own code contributions as contributions. They usually have no write access to the repository. Their contributions are reviewed by the project's maintainers and added to the repository once the quality typical for the project has been achieved.

At the top of the pyramid are the maintainers or committers of a project. In this role, a developer has a higher level of responsibility. This manifests itself, for example, in the fact that he or she can accept contributions. This right is often expressed through write access to the repository. At this level, control over the software, its quality and functionality takes place. In more complex projects, this level can be extended even further. One well-known example is the Linux kernel, in which there are subsystem maintainers at several levels and ultimately, with Linus Torvalds, only a single developer takes over the patches in the project repository. Another example is Eclipse Foundation projects, in which there is typically another project lead who has additional rights in the context of the Eclipse Foundation development process; for example, he or she can trigger the release process or initiate the formal election of committers or project leads.

4.1.2 Projects with informal governance

Such a contribution pyramid can arise chaotically or in a self- organised way. Many open source projects are started by one person as the inventor. At least in the first growth, such a person will keep the decisive role at the top of the project. In some projects, the person remains in this role as a benevolent dictator for life.

In long-lasting and growing projects, however, there are often generational changes. In these cases, new people come to the top who primarily distinguish themselves through their commitment and the extent of their contribution.

Such projects usually manage very well and for a long time without formal governance. A public source code repository and a mailing list suffice as a constituent element.

In terms of participation, these projects certainly pose the greatest challenge from a company perspective. In order to gain any acceptance at all and possibly later influence on such a project, competence and willingness to commit to the project on a long-term basis must first be demonstrated in practice. Because this always depends on individual persons in informally organised projects, this involves entrepreneurial risks.

The lack of legal unity and informal governance also makes a legal assessment of such a project difficult and thus the risk assessment of a contribution uncertain. For example, potential antitrust issues arise when collaborating with other companies in an unregulated environment.

Finally, projects of this type naturally lack formal arrangements for conflict and crisis. This need not be a problem, but it can have a negative impact on the longevity and dynamics of a project under certain circumstances.

4.1.3 Charter-based open source projects

Especially in the industrial context, there is often a need for more formal governance. For example, in vertical ecosystems, there are a number of projects and project families that define a form of membership through a charter. The idea of this approach is to ensure the commitment of the organisations interested in the project by linking the membership with an annual contribution or a commitment to provide development resources.Thus, the project has a base budget with which to drive development.

Membership is typically accompanied by control rights. Steering committees determine the use of the project budget and define working groups in which specific issues are advanced. Members have the right to participate in the various committees and thus have greater control over the project.

In this case, however, the projects are also open source projects, which means that in principle everyone has the right to participate in the form of contributions and can use the project freely according to the chosen licence. Often the project charters also explicitly open up possibilities to obtain further rights and influence through active participation in the project. In general, however, the strategic question for contributions to projects of this type is whether a possibly desired influence makes it necessary to become a member of the project. Only when the planned contributions exceed simple bug fixes and minimal patches is membership quickly advisable.

4.1.4 Foundation-based open source projects

From a series of successful projects, individual foundations have emerged from the previous model of control by a non-profit enterprise and have expanded their scope over the years. Examples of this are the Linux, Eclipse and Apache Foundations. These unite a large number of open source projects from different areas. They also often define a standardised governance framework that is established and facilitates the creation of a new project. Instead of setting up a completely new governance, for example in the form of an association, as in the previous example, established processes can be used in the foundations and complete governance parts can be outsourced through the foundation's support for the processes.

For a contributing company, this construct is certainly the most comfortable. The governance structure is standardised and a contribution to several projects of a foundation only requires a comprehensive legal review for the first project. The quality standards of the foundations ensure high process and software quality, so that a long-term commitment to an open source project typically entails low risks.

4.1.5 Openness of governance models

A company or a manufacturer that decides to place a software project under an open source licence obviously gives up a significant part of the control over its product.

Even if we see the dynamics with which existing open source projects are developing and the extent to which innovation is emerging as Open Source Software, this loss of control is perhaps perceived as a particular risk from a business perspective. It is therefore obvious to secure control over the open source project by constructing a governance model.

A long-term commitment to an open source project typically entails low risks.

Unlike the open source licences, which follow an open source definition that is tested and recognised by the OSI, there are no fixed rules yet for the governance models.

Therefore, before participating in an existing project, it should be checked, if necessary, whether the governance rules actually and sufficiently enable the character of a community-oriented joint development and cooperation. In the interest of creating a strong community process, this applies equally to the formulation of one's own rules when founding a new project.

Under the catchphrase »open governance«[27], [28] criteria can be identified that a project must fulfil in order for the openness determined for the code by the licence to be reflected in the governance model. In particular, it is open to new participants and it is oriented towards cooperation and the creative process.

An open governance model addresses issues such as

- Distributed or neutral ownership of the rights to use contributed code
- Neutral ownership of the infrastructure, such as the internet domain of the open source project, but also aspects such as naming and image rights to project resources.
- Licensing of project branding for, for example, proprietary products
- Transparency regarding
  - Decision-making processes
  - Code of Conduct and process for compliance
  - Management of the project budget, should one be levied (see charter-based open source projects).
  - Process for the appointment/retirement of maintainers/committers
  - Roadmap process
  - Release management

Open project governance is a quality feature.

Open project governance is a quality feature that ensures that individual community participants have a clearly defined influence on the project and that the processes in the project are transparent for all involved. Excessive influence should thus be avoided.

27 Meijer, Lips, Chen: Open Governance, ↗ https://www.frontiersin.org/articles/10.3389/frsc.2019.00003

28 See, for example, open governance models in the context of the Cloud Native Computing Foundation (CNCF): ↗ https://opengovernance.dev/

4.1.6 Split of open source projects

The open source licences guarantee all users and thus also all developers the freedom to offer and distribute their own versions of the software. In the event that a major contribution is not accepted by the management structure of an existing project mentioned above, this means that the modified software can be continued independently in a split-off project.

> Splitting up a community can invigorate innovation and bring new dynamism to a potentially cumbersome project.

But this step needs to be well thought out. Splitting a community can stimulate innovation and bring new dynamism to a possibly sluggish project. This was the case, for example, with the 386 community's split from Andrew Tanenbaum's Minix project. This gave rise to the Linux kernel.

On the other hand, such a split may also mean the disconnection of integration and security in the original project. The backporting and unification of such patches from the original project are possible to a certain extent, but they mean considerable additional work. Also, each split brings not inconsiderable uncertainty and friction losses.

4.1.7 Handling IP and copyrights

Another aspect of project governance is the handling of IP and copyright rights of the contributors. The IP rights, for example patents affected by the source code, are usually regulated by the open source licence used. Some licences have special patent clauses that usually grant a royalty-free right to use the patents involved. This is a relevant aspect when deciding to contribute to a project.

> An unresolved patent situation increases the risk when using the software and thus reduces he attractiveness of the project.

If a contribution concerns own patents, it is usually desirable to use a project that uses a licence with a patent clause. An unclear patent situation increases the risk of using the software and thus reduces the attractiveness of the project.

Formally, unrestricted use can be regulated by a Contributor Licence Agreement (CLA) or as an actual transfer of copyright to the entity by a Copyright Transfer Agreement. There are many examples for the concrete design of a CLA[29] in which, among other things, re-licensing as a proprietary product can be regulated.

29 ↗ https://en.wikipedia.org/wiki/Contributor_License_Agreement

4.1.7.1 Assignment of »copyright«: Individual or Entity

At least in the projects with informal governance, the contributor retains all copyrights. The code is thus made up of components with different owners, which is made usable by others through the common open source licence. One advantage of this approach is a lower barrier to entry, since in the best case no further arrangements need to be made in addition to the open source licence. Thus, the examination of a possible contribution is easier and less restricted. In addition, the spectrum of owners promotes the independence of the project; it becomes more difficult for individuals to dominate the project. The disadvantages of this approach are already described above in the section on informal governance. In particular, the licence can hardly be changed. Even jumping to a newer version of an open source licence is only possible with the consent of all copyright holders, unless explicitly regulated at an early stage. This is virtually impossible for a successful project after a few years.

In a written governance model, in contrast, it can be agreed that a contributor transfers the rights of use defined in copyright law to an entity behind the project for its free use (unrestricted republishing rights). Such an entity can be an individual company or a legal construct, such as a foundation or an association. Here there is a separation between the licence for incoming contributions (inbound) and the licence for the finished software (outbound). The advantage here is that the copyright belongs to a single strong entity (such as the Free Software Foundation), which can defend it against infringements and attacks if necessary. Even basic changes, such as updating the licence to a newer version, are no problem with this. A disadvantage of this procedure can arise from the fact that there is a dominant faction that can determine the project and the contributions of the contributor. This poses a risk, especially in company-dominated projects. A previously free software can also be transformed into a proprietary product here.

4.1.7.2Protection against third-party »copyright«

An open source project releases the use of the software it contains under the open source licence. A user must therefore be able to rely on the fact that all copyright holders of the project have also agreed to the use of their copyright under the licence. Incorrectly included content from a third party copyright holder can result in at least the parts containing that copyright not being usable. It is therefore in the interest of the open source project to ensure that each contributor only contributes content for which she owns the copyright or has an appropriate licence. This can be, for example, content from another open source project that has been released under a compatible licence.

To maximise the protection of these aspects, some open source projects make use of special agreements such as the Contributor Licence Agreement (CLA) mentioned above. Alternatively, there are lightweight mechanisms such as a Developer Certificate of Origin (DCO), in which the author of the contribution merely declares ownership or the right to redistribute the material. In the simplest case, the »inbound = outbound« rule applies, which ultimately accepts contributions based on the chosen open source licence; no further regulation takes place.

4.2 How can businesses participate in open source projects/ Contributions

The possibilities for active participation in open source development are as diverse as the use cases of software as a whole. Basically, participation includes all activities that go beyond the mere consumption of the software.

- This starts with the support of other users, for example on a mailing list.
- Feedback to the developers about bugs or change requests
- Qualified error messages possibly even with patches for correction
- Testing new versions
- Creation, extension and translation of documentation
- Packaging the software for specific distributions
- Implementing new methods, modules, features, etc.
- Review of code from other contributors
- up to the comprehensive responsibility for the parts of the project and initiating
- and driving new open source projects

All open source projects are at least partially interested in active participation by the communities.

All open source projects are at least partially interested in active participation by communities. Successful projects therefore show themselves to be inviting and open, even if they primarily pursue a company-driven development model.

At the same time, however, successful open source projects usually have a management and control structure and there are guidelines for the form and content of the collaboration. Such a structure can be determined from a company-driven development model as described above.

However, it may also be democratically constituted within the framework of a foundation, for example, or it may have grown informally in the form of a meritocracy. Especially in the projects described above with informal governance, it is advisable to seek agreement with the project before making and contributing larger contributions.

4.2.1 Open Source Participation from an Economic Perspective

In order to avoid the above-mentioned additional work involved in maintaining a separate branch of an open source project, there are good economic reasons for also making substantial contributions to the original project and thus making it available to the general public.

Even small contributions have an economic benefit for the contributing companies. Feedback to the developers serves to motivate them and, at best, can steer the development in a desired direction.

By contributing patches to the upstream community, not only is the effort of maintaining a separate software branch saved, into which the upstream patches then have to be ported back. Often, the ideas realised in such a contribution are also taken up by the community and developed further independently. In the ideal case, a small impulse is enough to initiate major changes.

There are other economically calculable values and benefits in participating in open source projects.

- In the corporate »beauty contest« for the best talent, active participation in open source projects and an open source contribution policy in the company is an attractive differentiator.
- Supporting other users can underline one's own competence as a service provider in the domain of an open source application.
- Open source reference implementations can be used to set standards, create interoperability and generate trust between otherwise competing parties.

In our complex global economy, the constantly changing information and data processing requirements of the companies involved cannot be adequately met either by standard software from traditional providers or by in-house development.

Digital transformation and thus future viability can only succeed through collaboration and cooperation with open source methodology and Open Source Software.

4.3 Collaboration Tooling

Open source development is characterised by the cooperation of otherwise organisationally unconnected individuals over large spatial and thus temporal distances. In this setting, technical collaboration tools play an important role. Giving concrete examples or even recommendations for tools is beyond the scope of this guide. Every project puts together its own toolbox. Nevertheless, we would like to share a few basic considerations and experiences here.

4.3.1 Communication

Asynchronous and at the same time fluid and comprehensible communication is essential for collaboration. The larger and more successful a project becomes, the more participants need to be involved in communication, often across multiple time zones.

Asynchronous and at the same time fluid and comprehensible communication is essential for collaboration.

Email (especially in the form of mailing lists) and chat are used for communication as a matter of course. Typically, the relevant open source projects also have a website (homepage) on which the project presents itself and offers further information, for example in the form of a wiki, a blog or a discussion forum for users.

This is important for the planning of a new open source project, as the effort required, for example, for the moderation of a mailing list or the maintenance of a project page is not insignificant, but is often forgotten in the initial estimate.

4.3.2 Tool chain

Since open source development is essentially about software source code, the version control system (VCS) plays a central role. In addition to the traceability and retraceability of changes, this also involves the coordination of parallel development threads (branch, fork, merge, etc.). In addition to historically existing VCSs such as CVS, RCS, SCCS and Subversion (SVN), there are a large number of proprietary offerings and there is Git.

The choice of VCS is relevant insofar as a change is typically accompanied by the loss of metadata and history. For current and especially for new open source projects, Git is probably the general standard as a tool and data format. Git is a distributed system and there are several well-established platforms that offer central Git servers for open source projects free of charge on the internet.

Automation and integration: Modern software development uses a variety of tools, methods and integrated environments with which operational elements of the creative process can be simplified or completely automated.

Examples of such tools are IDE, Module Library, Artefact Repository, Build Pipeline, Test Automation and many more.

The concrete design of such a development environment depends, among other things, on the chosen programming language and the deployment and operating format of the resulting software. For a classically compiled C program, the environment looks different than for a J2E Java application or for a containerised Go microservice. Powerful open source tools are available for all tasks and areas. A deeper look at the many options for designing such development environments is beyond the scope of this guide.

> Tools can support the social character of interaction, they cannot replace it.

For each individual open source project, it makes sense in any case to describe the prototypical development environment together with the coding standards.

Platforms: When selecting a suitable platform, it is also a question of role and rights management (commit, review + merge, comment, etc.), the management of issues, the automation of workflow and the integration into various development environments.

With control over these functions comes structural power over essential processes in the project. Tools can support the social character of interaction, they cannot replace it.

The use of even free offers for central source code repositories and version control systems has a price. This means that the question of who pays this price and how sustainable and secure this business model is is also part of the considerations for selecting the appropriate tool.

4.3.3 Creativity

Beyond the purposes mentioned, there is a wide field of technology to support creative collaboration. A good overview of creative methods for teams can be found in the ↗ Open Practice Library.

And there are tools like diagram editors, whiteboards, Kanban boards, ↗ Liquid Democracy to apply these methods and support the creative process.

4.4 Conclusion

The models and ways of contributing shown here come from very different situations and are driven by a wide variety of intentions and motives.

Open Source Software is used everywhere: in smartphones, as web services or as infrastructure in the cloud. Even in proprietary software products, some components, libraries, tools or frameworks from the open source world are almost always used. In accordance with the image of the contribution pyramid described above, all companies should at least become aware of their role as users of Open Source Software. Actively shaping this role and, for example, explicitly allowing active participation in communities with experience reports, bug reports or small patches is a first step.

It is also important as a user to know and exercise the active rights of Open Source Software.

It is also important as a user to know and exercise the active rights of Open Source Software. Where much of the underlying technology and components are already available as open source in proprietary software today, it may make sense to actively engage in a collaborative development process and increase the portfolio of open source solutions.

When creating a new open source project, as described above, there are many degrees of freedom that, if chosen correctly, can have a strong influence on the success of the project. Especially a well- chosen project governance, for example the use of a proven open governance approach or a methodology specified by a foundation, makes it easier for interested companies to participate in a project. This results from the fact that the mechanisms are established and proven. The risks are thus much easier to assess. In addition, they are known to the reviewing lawyers, and may even have already been reviewed. This reduces the risk of participating in the project and makes it attractive.

5 Business models around Open Source Software

A basic principle of Open Source Software is that no licence fees may be charged for its use. This condition is anchored in the OSI's open source definition. Beyond that, however – and this is the crucial point – **any remuneration agreement for any conceivable service for and with Open Source Software is permissible**. Software that may not be used commercially by licence is not Open Source Software according to the definition of the Open Source Initiative (OSI).

Whatever business model one comes up with, one aspect distinguishes the commercial use of Open Source Software: Only indirectly can a unique selling proposition be built up with it. The accessibility of Open Source Software and the freedom rights associated with it facilitate the emergence of alternatives. Every person who works with Open Source Software for business purposes should provide the software on which his or her service is based to his or her customers in the full knowledge that he or she as a supplier – in terms of the software – is easily replaceable. Accordingly, the added value is provided in awareness of the suppliers own special competence and the unique selling proposition is formed from this. The use of Open Source Software can be a sales-promoting advantage: open source services give customers a greater degree of freedom in their relationship with their supplier. Seen in this light, Open Source Software has always been and still is a special instrument in a market economy.

Open Source Software is therefore not a business model in itself, but »only« a special, cooperative development method. Thus, the development of the scene is accompanied throughout by the question of what exactly such business models can look like with an orientation towards open source. For a long time, for example, companies doubted whether professional commercial support was even possible for »free« software. It was also questionable at first whether it could make economic sense at all to support Open Source Software beyond pure use in the form of development contracts or through one's own participation. Conversely, open source communities doubted whether companies would abide by the rules of the game of free software and whether commercial exploitation would be compatible with the spirit of the communities.

There is no longer any question of this today. Open Source Software is an integral part of many commercial products and, especially in the area of cloud services, open source is the only way to implement the scalability that necessarily characterises such services in an economically sensible way. Spectacular takeovers and valuations of companies that base their business model primarily on the development of Open Source Software confirm the economic importance of the concept of open source.

The time when open source was a special topic and presented at separate trade fairs, for example, is over. Today, open source is implicitly or explicitly part of all areas of the economy where software is relevant. This is also reflected in the organisation in associations: For example, there is the Open Source Business Alliance (OSBA)[30] and also within Bitkom e.V. there is the established Open Source working group[31].

The time when open source was a special topic and was presented at eparate trade fairs, for example, is over.

30 cf. ↗ https://www.osb-alliance.de/
31 cf. ↗ https://www.bitkom.org/Bitkom/Organisation/Gremien/Open- Source.html

The **existing business models** can be **classified** in terms of whether the business model uses the Open Source Software as a means to achieve an independent business purpose (business models **with** Open Source Software) or whether the business model is built directly around the Open Source Software (business models **for** Open Source Software). Following this classification, examples of business models that can be observed on the market are described in the sub-chapters. This list does not claim to be complete, but gives a comprehensive picture of the mainly observable models.

5.1 Business models with Open Source Software

In these business models, Open Source Software is used to develop a product or provide a service. The Open Source Software fulfils a means to an end here and is often only consumed. However, many companies also use the mechanism to specifically develop non-competitive areas of their software stack more efficiently through cooperation with other companies or to improve quality through a broader user base.

5.1.1 Services with Open Source Software

In this model, services are offered that are provided through the use of Open Source Software, also known as software-as-a-service (SaaS). Social networks, web shops, search engines, cloud providers, they are all operated using Open Source Software, sometimes on a massive scale. Customers pay for the operation of the service, or the services are financed through business models such as the sale of advertising. Whether this is done with open-source or proprietary software is irrelevant to the actual service. Open Source Software flows into the business model as a favourable cost factor.

Last but not least, the availability of Open Source Software has contributed massively to offering start-ups the opportunity to access many highly powerful software stacks with minimal investment and to realise innovative services at an unprecedented speed.

5.1.2 Open Source Software as a Service

A special form of service with Open Source Software is the operation of Open Source Software as a service on the Internet. Here, a distinction can be made between operation by the manufacturer or by third parties.

Operation by the manufacturer is, for example, Open Source Software that is made available or mainly developed by one company. In addition to the freely available software, however, this company also offers an internet service that releases users from the installation and maintenance of the software and makes it easily available to them.

Examples exist in the area of databases or virtual communication platforms and, for example, in projects of the Cloud Native Computing Foundation[32].

Especially for Open Source Software, which has achieved a high level of distribution due to its free availability, offering it as a service is often an attractive form of generating income that finances the maintenance and further development of the software. However, this model then falls more into the category of »business models for Open Source Software«.

In third-party operation, the software is provided by a company that is not the primary or sole developer. An example of this is the operation of services by cloud providers. There has been some controversy around this model in recent years, as some manufacturers have seen themselves at a disadvantage to the usually much larger cloud providers and have tried to compensate for this disadvantage by changing their licensing. The licences to which these vendors have switched typically exclude use by competitors in the operation of the software and are thus no longer open source licences. Open source licences require non-discriminatory usability, even by competitors. One must also note that the major cloud providers are among the largest open source contributors.[33]

Open source licences require non-discriminatory usability, also by competitors.

5.1.3 Products with Open Source Software

Comparable to services, Open Source Software can of course also incorporated into the development of products. Here, open source components are built into the product and often represent a large share of the product's software. This is the classic use case, Open Source Software is used in a software context and redistributed with the product. An obvious indication of the relevance of this model are the lists of open source licences that can be found in many configuration menus or manuals of products such as televisions, routers, computer games and the like.

5.1.4 Open Source Software as enabler for other business models

In this business model, Open Source Software is used to facilitate the use of a service or the delivery of another business purpose. Companies with this model create quite extensive Open Source Software. This can be, for example, an operating system for smartphones, a browser, or development tools. The widespread use of this software can then support such different business models as the collection of data for advertising or the increased use of provided, fee- based cloud services. The investment in Open Source Software therefore only indirectly serves the business purpose, but offers control points by means of which one's own business model can be advanced.

32 see many CNCF Cloud Native Interactive Landscape projects, cf. ↗ https://landscape.cncf.io/
33 cf. ↗ https://opensourceindex.io/

5.2 Risk assessment with regard to the use of Open Source Software

If we look at the business models described, the use of Open Source Software made available is in the foreground. A provider benefits from the development of the open source community, but also relies to a large extent on the community developing the software in the intended sense. This represents a risk that must be assessed by the consuming company. Particularly in the case of a large dependence on individual open source components, a consuming company must install measures to control the risk.

On the other hand, today's market dynamics, which have also arisen due to the open source mechanism, force a rethinking of the strategy of how non-competitive parts of one's own product or service portfolio are developed and made available over the entire life cycle. The risk then is that sticking too long to proprietary in-house development ties up too many developers and the company's speed of innovation suffers. Risk control involves increased investment in Open Source Software, for example by participating in or even founding open source communities.

If a company wants to participate more in open source projects, two options are available. One is to commission third parties who then actively participate in open source communities. The matching business models are described in the next subchapter. This section describes the options for direct participation in open source projects.

At this point it is useful to distinguish between more general platform software and vertical software. Platform software is typically not domain-specific, but of a general nature. Operating systems that are needed to operate a device but do not provide any functionality that is part of the company's core competence serve as an example. In contrast, vertical Open Source Software as a product or service component is about developing typical but non-competitive software in a specific market in a joint collaboration of market participants. Examples of this form are the Networking Working Group of the Linux Foundation[34] in the field of telecommunications networks or the Academy Software Foundation[35] in the field of motion pictures.

> Especially in the case of a large dependency on individual open source components, a consuming company must install risk control measures.

34 cf. ↗ https://www.lfnetworking.org
35 cf. ↗ https://www.aswf.io

5.2.1 Participation in platform Open Source Software

For general software projects, a typical strategy is to participate sporadically at first. This can take the form of bug fixing, smaller feature contributions or just community activities such as bug reporting and participation in discussions. Depending on the risk assessment, the aim here is usually to fix urgent defects in the software from the perspective of one's own use. However, the example of operating systems, and here in particular the example of Linux, shows very well that participation can quickly go deeper. For companies that provide their own hardware such as sensors or actuators, there is a need to make their hardware usable for Linux by means of drivers. In Linux, drivers are part of the Linux kernel so that it does not become incompatible when the kernel is further developed and is easily available to the users of the hardware. Consequently, it is obvious for hardware manufacturers to contribute software to the Linux kernel.

For most companies, this form of collaboration with open source communities is driven by getting involved in established ecosystems and supporting existing standardisation through the Open Source Software. Rarely is it about driving new standardisation through Open Source Software at this general level. But this is not far-fetched either, as an example Zephyr[36], an operating system for low-resource devices, which has only emerged from industry collaboration in recent years and is supported by a large number of companies from different sectors.

5.2.2 Participation in vertical open source projects

For many companies, this participation will be an interesting strategic model to release development resources from non-competitive software parts. This enables them to concentrate on diversifying software parts and thus strengthen their market position.

The idea of vertical open source projects is that software that is currently developed proprietarily because it is necessary for the operation of the business model must also be provided in this or a slightly modified form by other market participants. In this case, there is the potential to jointly develop these software components in cooperation with other market participants and in this way standardise them for the industry. Through joint development, one's own investment in the software can be reduced in the medium term and thus the desired effect of releasing developers can be achieved.

36 cf. ↗ https://www.zephyrproject.org

The examples mentioned above show very well the motivation for cooperation. In the telecommunications sector, it is about a highly reliable provision of the network. The quality of the hardware and software used is more important than extensive diversification possibilities, ideal for cooperation in the open source field. Even in the motion picture environment, diversification comes from the creativity of those involved. The provision of the tools is more of an obstacle because they mean a high investment that is often hardly affordable for the companies involved, but on the other hand the efficiency of usability is in the foreground, i.e. the efficiency of the creatives.

5.3 Services for Open Source Software

The use of Open Source Software is contrasted with business models in which services for the adequate operation of Open Source Software are remunerated. The companies offering these services do not necessarily have to develop Open Source Software themselves. For providers of their own Open Source Software (application providers), additional services can be a secondary profit model. It can be observed that a wide variety of billing metrics are generally used in these business models. Examples are contingent billing or usage- based subscriptions. The services can be typified as follows:

5.3.1 Support

This includes both technical and non-technical customer support, usually realised through a helpdesk or call centre. Support service providers may specialise in the technical aspects of installing and integrating new software into existing systems, but may also provide assistance with day-to-day application problems.

Support contracts are often made that specify different support levels, consulting availability, and response times. Support levels include first, second and third level support, each level representing a further escalation within the system. There is typically also variation in billing. The range goes from individual invoicing of support requests to subscriptions that allow support requests for the agreed contract period.

5.3.2 Development

In the case of customised contract development, the service provider integrates certain functions or design features into the Open Source Software. In this way, for example, independent applications are created on the basis of an already existing software version, which are tailored to the client. Such an offer for Open Source Software also makes it possible to realise special features (earlier) that are needed for a service with Open Source Software independently of, for example, a community-driven roadmap. Often, but not necessarily, these custom developments are published under an open source licence and can find access into the original open source project if there is sufficient interest.

In relation to the »Open Source Software« model, the service may not only consist of the pure development of the software, but also of organising contact with the open source community, cooperation or the back-publication of development statuses.

5.3.3 Operation and provisioning

In contrast to the Open Source Software-as-a-service model, this business model is about preparing the Open Source Software for a specific customer in such a way that the customer can use the software directly. This includes services such as the operation of a service on customer-specific hardware, integration of the software, and provisioning using the customer-specific software management mechanism to make the software available on the customer's computers. In this model, the configuration of the software for the customer is also done by the suppliers. This model is typically combined with other models from this subchapter.

5.3.4 Maintenance

Maintenance of software in use is usually limited to a certain period of time (e.g. one year) and to a defined scope of services (such as the installation of regular security updates). Software that is no longer actively developed is often still used in complex IT architectures and must be maintained. Even after the official discontinuation of the open source project, these tasks can be taken over by third-party providers in order to keep a software or infrastructure running and to extend the period of use.

5.3.5 Consulting

Consultancy services can cover all points of the life cycle, for example studies, analyses and conceptions. They are aimed both at the phase before the introduction of software (such as selection and evaluation or tendering) and at accompanying it in order to successfully shape the transition period as well as its later use and to adapt corporate processes (for example, creation and implementation of a security concept). There are also consultants who support companies in their own software projects. This can be useful when Open Source Software is to be published for the first time and when dealing with open source communities or licensing conditions.

5.3.6 Certification

Another business model is based on the fact that Open Source Software usually comes without guarantees. As Open Source Software moves into more and more areas, secondary properties such as safety or security become more important. In domains such as the automotive industry or aviation, there are ISO standards whose compliance must be proven by the software in order to obtain approval for the operation of the product including its software.

> As Open Source Software penetrates more and more areas, secondary properties such as safety or security become more important.

In this business model, services are offered that carry out a qualification of the Open Source Software according to one of the ISO standards for a specific use case. This certification enables the usability of Open Source Software in corresponding areas. As an example at this point, the ELISA Project of the Linux Foundation[37] should be mentioned, which is about the use of Linux in safety-critical applications. This project is about processes and methods to enable such use. This subsequently forms the possibility of offering certification services on the basis of the standardised processes.

5.3.7 Training

Training courses complete the offer around Open Source Software. Numerous providers – mainly small and medium-sized enterprises (SMEs) or individuals – have specialised in offering a wide range of courses and certifications. The target group includes users as well as administrators and programmers. Companies can train their employees in a targeted and professionally supervised manner and thus increase their internal know-how. Large Open Source Software providers also include training courses in their repertoire: in this way, they disseminate and broaden knowledge about their open source offerings and at the same time offer service to customers.

5.3.8 Dual licensing

In principle, there are many variants of dual licensing[38]. In this chapter, only the combination of open source and proprietary licences will be considered. In this case, software is made available under a typically rather restrictive open source licence. If the users of the software cannot or do not want to use it under the conditions of the open source licence, they can acquire the necessary rights of use in a proprietary version for a fee.

This can make sense for both parties: The software customer does not have to fulfil the open source obligations normally associated with the software, for example a

37 cf. ↗ https://elisa.tech

38 In principle, this type of licensing also exists purely internally within open source: In this case, the developers of a software are interested in ensuring special security with regard to licence compatibility through multiple licensing under different open source ↗ licences.

copyleft effect. The software producer, on the other hand, can charge the users directly for special features whose development it cannot cover through the sale of services.

A special case of dual licensing is the open core model. Here, the complete software is not made available under an open source or proprietary licence, depending on the version, but part of the software is distributed under an OSI-certified open source licence and another part under a proprietary licence.

Often, a generally usable basic version is available as Open Source Software and additional components, for example for integration into an enterprise environment, can be acquired as proprietary software against payment of licence fees. One challenge of this model is that the decision as to which component goes into which version is often not an easy one and, as a matter of principle, puts the other user group at a disadvantage.

5.4 Further models

In addition to the »classic« business models described above, there are a large number of other models, of which we will briefly highlight only a small number here.

5.4.1 Donation-based financing model

Many open source developers work on a voluntary basis and do not create their software for a company or in any other business context, but as a voluntary service. The motivation for this can be, for example, curiosity, learning, reputation, fun in experimenting or involvement in a community.

Some users would also like to make a monetary contribution to such volunteers. Especially in projects that have become more widespread and are also used commercially, it may also be in the interest of companies that use the software to enable more sustainable development through financial support.

A donation-based funding model is a good way to do this. This can be done, for example, through a foundation that acts as a financial proxy for a volunteer community, or through platforms that allow donations to be made to specific projects or developers. There are now many variants of this, be it crowd-funding, subscription or integrated into a code-hosting platform.

Many open source developers work on a voluntary basis and do not create their software for a company or in any other business context, but as a voluntary service.

5.4.2 Foundation Model

The large foundations offer a special cooperation model. Through their legal form and policies, they enable companies to work together openly and independently, which protects them from antitrust problems. As a rule, open source projects or working groups are created in this way, which are governed by their own community contract. These contracts typically include a membership fee that must be paid by the participating companies and a steering organisation that serves to distribute the budget collected through membership fees among the projects.

In principle, no one can be prevented from participating in an open source project, but the contractually regulated structure ensures sensible budgeting for the activity and a goal-oriented approach. This model is typically used in the vertical open source activities of the participating companies described above to legally anchor the cooperation.

Another business model practised by the foundations is the organisation of conferences and user meetings. The aim here is to offer the open source communities platforms for exchange and to bring users together with contributors.

6 Strategic Consideration of Open Source

6.1 Open Source Software in the Company

Just 15 years ago, the use of Open Source Software in companies was considered revolutionary. Companies that publicly declared that they were using Open Source Software were observed with suspicion and mistrust. Today, on the other hand, all studies[39] consistently show that the use of Open Source Software in companies has become quite normal, in fact it is almost unthinkable to do without it.

There is probably no company left that does not use Open Source Software. It is therefore all the more astonishing that Open Source Software is still rather reduced to technical and legal issues in many companies, and is not regarded as a new philosophy and collaborative production, distribution and business model. The fact that Open Source Software has been revolutionising ICT ecosystems for several years shows that this does not do justice to the importance of Open Source Software. This often happens quietly and unnoticed at first – the effects of this revolution are nevertheless far-reaching and obvious.

The strategic and economic importance of Open Source Software is shown by developments in recent years, such as the dominance of the Android mobile operating system, the market penetration of Open Source Software frameworks for artificial intelligence, investments and acquisitions of companies such as Redhat or Github by IBM and Microsoft. Companies should consider all the influencing factors of Open Source Software and regularly review whether or not their attitude and strategy is conducive to business. Manifesting this in an open source strategy is recommended.

> The strategic and economic importance of the topic of Open Source Software is demonstrated by developments in recent years.

39 See for example ↗ https://www.bitkom.org/opensourcemonitor

6.2 Open Source Strategy Development in the Company

6.2.1 Basic Consideration of an Open Source Strategy

The following considerations are intended to make it easier for companies to define an open source strategy and to provide assistance in taking the necessary accompanying measures. Whether all the above-mentioned considerations and analyses are carried out or are present in the companies depends very much on the size of the company. The starting point for consideration may be a common definitions of the term »strategy«:

»Strategy is defined as 'the determination of the basic long-term goals of an enterprise, and the adoption of courses of action and the allocation of resources necessary for carrying out these goals.' Strategies are established to set direction, focus effort, define or clarify the organization, and provide consistency or guidance in response to the environment«[40]

With regard to Open Source Software, it must be determined to what extent its use, collaboration in or publication of own open source projects can contribute to achieving the business goals. In order to derive such a strategy, various aspects need to be considered.

The vision, goals and general strategy of the company are the essential starting point for defining a strategy in the field of Open Source Software. Depending on this, the use of Open Source Software should be planned as a means of achieving the company's goals and a congruent open source strategy should be established. A different form depending on the product, segment, division, unit or company of a company may be opportune or even expedient depending on the size and diversity. The use of Open Source Software should never be seen as an end in itself, but as a supporting means to achieve the company's goals. Consequently, the open source strategy of a company is always subordinate to the corporate strategy.

40 Cf. the general definition in Wikipedia, ↗ https://en.wikipedia.org/wiki/Strategic_management.

6.2.2 Strategic directions

An efficient and effective use of Open Source Software can be divided into four essential maxims of action and thus open source strategies:

1. Prohibition of Open Source Software as far as possible (Limit Open Source Software)
2. Promoting the use of Open Source Software (Use)
3. Promoting the contribution to Open Source Software (Contribute)
4. Promoting the creation of Open Source Software (Create)

These four strategic directions should not be understood as self- contained and delimited, independent concepts. Rather, it can be seen that depending on the company and the maturity of the topic of Open Source Software as well as the individual aspirations and the expected benefits, the transitions between these strategies are fluid. Starting with unmanaged use or suppression of Open Source Software, through conscious use (Use) and participation (Contribute) to the independent creation of Open Source Software (Create).

6.2.3 Goals of an open source strategy

Drivers and influencing factors for the definition of an open source strategy can be, among others:

- Setting standards
- Market presence and external impact
- Gaining market share
- Retaining customers and suppliers
- Talent acquisition
- Knowledge acquisition
- Cost reduction
- Safety requirements
- Market and customer requirements

6.2.4 Résumé and necessity of an open-source strategy

The selection and consistent specification of an open source strategy and the extent and scope within a company may depend on a wide variety of factors. However, it is certain that no company can do without Open Source Software. In this respect, in order to ensure the desired exploitation of the potential of Open Source Software as well as the possibly necessary reduction of risks that can arise from Open Source Software, a strategy and the corresponding measures for action should be defined. Against this background, the fact that more than 70% of companies do not yet have an open source strategy is all the more remarkable.[41]

At least a minimal open-source strategy is necessary to ensure that licence conditions are adhered to. For more details, see ↗ Chapter 7 »Compliance«.

When participating in open source projects and founding your own open source projects, further strategic considerations are necessary to answer questions of cooperation with other companies, governance for projects or community management. For more details, see ↗ Chapter 4 »Creating Open- Source-Software«.

41 See for example ↗ https://www.bitkom.org/opensourcemonitor

6.3 Open Source Program Office (OSPO)

As part of an open source strategy, it is important to determine how the topic of open source and the associated tasks are mapped out organisationally in the company. The establishment of an Open Source Programme Office (OSPO) has become established as best practice.

An OSPO is a central team that takes care of the different aspects around Open Source Software within a company in the sense of a holistic approach. Originally, this approach was mainly used by IT companies. However, due to the increased importance and spread of Open Source Software, it can also be useful and attractive for non-IT companies that are faced with the challenge of managing Open Source Software in the company and introducing corresponding processes and tools.

OSPO is the term most commonly used internationally, often a similar form of organisation is also implemented under other names.

6.3.1 Tasks of an OSPO

The responsibilities of an OSPO typically include the following tasks:

- **Open Source Strategy**
  The question here is how Open Source Software – its use, participation in, but also the initiation of new open source projects – can support the corporate and product strategies as well as help to achieve the corporate goals. In addition to the creation, the internal (and, if necessary, external) communication of the open source strategy in cooperation with the relevant stakeholders is also part of the OSPO's tasks.

- **Open Source Policy**
  When dealing with open source, rules must be defined and anchored within the company. This concerns in particular the use of Open Source Software within the own organisation or the own products, the participation in open source projects, but also the starting of own projects. It is the task of the OSPO to create and communicate the open source policy for the company and to provide appropriate training for the employees.

- **Open Source Processes**
  The definition, implementation, execution and continuous further development of processes for the management of Open Source Software in the company is also part of the tasks of an OSPO. For example, processes that help ensure licence compliance.

- **Tools**
  Due to the sheer mass of Open Source Software in companies, the associated processes can no longer be carried out efficiently without automation and tool support, especially in larger companies. The task of an OSPO is to introduce, administer and possibly also develop appropriate tools for the most extensive automation of open source management processes.

- **Training**
  Dealing with Open Source Software makes it necessary for the relevant employees to build up appropriate knowledge. This concerns the use of Open Source Software in compliance with compliance and security rules, but also the participation in open source working groups as well as the initiation of open source projects and the associated building of communities. The OSPO is typically responsible for developing, providing and continuously developing an appropriate training programme for staff.

- **Communication**
  Another task of an OSPO is to create transparency about a company's open source commitment, both internally for its own employees and externally for the interested public, developer communities, partners and customers. Especially when own open source projects are started, it is important to make them known (for example via blogposts, presentations at conferences or publications in the relevant media). Likewise, it is usually desirable to build and manage communities around these projects. An OSPO can advise and support development teams in these tasks.

- **Memberships in Open Source Associations and Committees**
  In the open source sector, influence is typically exercised through participation in communities. These can be project-specific communities, but also larger organisations such as open source foundations. An OSPO typically has the task of aligning memberships and engagements in these organisations with the strategic goals that a company is pursuing with open source. Such engagements need to be launched and supported according to the company's strategic goals.

The abundance of tasks of an OSPO is therefore large. To prevent it from becoming a central bottleneck, it is important to weigh up which processes need to be run centrally via the OSPO and which decisions should be made decentrally in the development departments. Tools for automating and supporting processes as well as self-services for employees play an important role here.

6.3.2 Organisational Aspects of an Open Source Program Office

There is no universal blueprint for the structure and organisational anchoring of an OSPO in a company. Rather, these depend on the company structure, the use as well as the strategic importance of Open Source Software for the company. OSPOs often start small, with only a few positions, sometimes even only one person.

On the one hand, there are OSPOs that consist entirely of full-time staff, and on the other hand, there are those that are completely virtual. In this case, they consist exclusively of part-time employees who carry out the OSPO work alongside their work in their actual department. Due to the diversity of its tasks, an OSPO has to work with many different parts of the company, such as:

- Development departments
- Product management
- Strategy departments
- Licence management
- Legal, IP, compliance and security departments
- Training departments
- Corporate communications and (technical) marketing
- Internal IT department
- Software procurement (purchasing)
- Personnel department

Therefore, a completely virtual organisation consisting exclusively of part-time employees from several of the above-mentioned departments can make perfect sense. Mixed forms are also conceivable. For example, there could be a core team of full-time staff supported by part-time staff from other departments in a virtual approach.

The possibilities for anchoring an OSPO in the company are equally diverse. On the one hand, there can be a central OSPO that is responsible for the entire company. If, however, the company consists of parts that act rather independently of each other, it may be advisable to set up decentralised OSPOs in the individual parts of the company instead of a central one.

Often an OSPO is placed in the reporting line of the *Chief Technology Officer (CTO)* or *Chief Operating Officer (COO)*.

Further information on the subject:

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

6.4 Open Source Foundations

There are many aspects to consider when getting involved in open source projects, whether as a contributor to existing projects or founding new ones: Legal frameworks, models of cooperation, successful community management and more. For a single company, these questions are often difficult to answer, as the necessary expertise is not always available in-house and the dependence on a single company is a hurdle, especially for building a community.

Open source foundations[42] represent a solution here. These are organisations which, as an association of several companies and often also individual developers, offer a neutral basis for cooperation on open source projects. The American ↗ Linux Foundation and the European ↗ Eclipse Foundation are examples of large, strongly industry-oriented organisations. The ↗ Apache Software Foundation is an example of a more community-oriented organisation. However, there are also a large number of other organisations43, in particular so- called »user-led foundations«44, which are driven by companies that act primarily as users rather than producers of software, or industry-specific foundations, such as the ↗ Academy Software Foundation for the film industry and ↗ Eclipse Automotive for the automotive industry.

An important function of foundations is that they provide an open collaboration model, often referred to as »open governance«.

An important function of foundations is that they provide an open collaboration model, often referred to as »open governance«. This is necessary because while open source licences guarantee software users extensive freedom, they say nothing about how that software is created. For a sustainable project, it is important to establish transparency about decision-making processes, to define a clear path for participation that promotes shared development, and to have a neutral body that carries key aspects, such as trademarks or project infrastructure. This is achieved through an open governance model in a foundation.

Foundations also define legal frameworks that enable cooperation between companies without the need to negotiate and set up specific new structures. This also covers, for example, antitrust requirements. Finally, foundations offer a valuable resource for sharing, learning or representing common interests. Since the challenges of dealing with Open Source Software are very similar for many companies, a joint approach is particularly appropriate here. In this respect, targeted participation in foundations can be an important part of the open source strategy.

42 The term »foundations« has become a generalised term in this context. Behind it are various forms of organisations that work without the intention of making a profit. In Germany, this is often the »Verein«. In Europe, it is often the Belgian AISBL. In America, so-called 501(c) ↗ organisations are common.

43 The community-maintained Foundation Directory provides a good insight. However, it does not claim to be complete. ↗ https://flossfoundations.org/foundation-directory/

44 See for example ↗ https://oss.cs.fau.de/2020/03/12/research-paper- the-ecosystem-of-openkonsequenz-a-user-led- open-source- foundation-oss-2020/.

6.5 InnerSource

»InnerSource« describes the procedure of applying open source principles to software development within a company without publishing the programme code outside the company.

This makes it possible to benefit from the advantages of open source development models without making development public and without abandoning the business model of selling software licences. This includes making source code available within the company across team and project boundaries and also accepting contributions from people who are not members of the actual team.

InnerSource can thus lead to improved collaboration and increase development speed and quality, as well as help to reduce costs. Multiple developments can be avoided and teams depending on projects that do not have enough development capacity can contribute needed features themselves. This is particularly successful in situations where different parts of the company need software components with similar requirements, for example internal standard libraries, infrastructure components or development, test and deployment tools.

It is important to note that InnerSource is not the same as Open Source. InnerSource code is still proprietary code and the dynamics of an InnerSource project will be different from those of a publicly, in many cases volunteer-led, project. Nevertheless, many concepts can be transferred and positive cultural changes can be achieved.

There are many examples of InnerSource initiatives. The book ↗ Adopting InnerSource, Principles and Case Studies describes in a series of case studies the successful application of InnerSource in companies like Bosch, Ericsson or Paypal. More material can be found in the ↗ InnerSource Commons Community. These include a series of InnerSource ↗ Patterns which capture tactics for using InnerSource.

7 Open source compliance

According to Wikipedia, »compliance« means »adherence to a rule, such as a specification, guideline, standard or law«. Wikipedia further explains: »Regulatory compliance describes the goal that organizations strive for by ensuring that they are aware of the relevant laws, guidelines and regulations and take measures to comply with them.«[45]

↗ »Compliance« is an English word. The Duden dictionary from 2009 does not yet know it.[46] Its current online version says that the business world uses it to mean »compliant, prescribed, ethically correct behavior«[47]. And Wikipedia states that »Compliance [...] is the business and legal term for the adherence to rules (also conformity to rules) of companies [sei], i.e. compliance with laws, guidelines and voluntary codes [meine]«[48], while »in the legal field [...] the term compliance basically (describes) the observance of rules in the form of law and order«[49].

Compliance therefore refers to an activity: it is about the deliberate and planned adherence to requirements, not a kind of »passive« adherence to the law. In the case of *Open Source Software*, this difference is crucial: as a rule, people are not allowed to use this type of software without first making a contribution.

Permission to use and action are linked via the open source licenses. These have a typical structure: firstly, the users of the software are granted exploitation rights. Secondly, this permission is subject to conditions. This can be illustrated using the example of the simplest open source license, the MIT license[50]. It begins with a *copyright line* and the following statement:

»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, [...]«

The following postscript ***»subject to the following conditions«*** binds the permission to this condition:

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

This actually means that MIT-licensed software may only be distributed if all »substantial portions of the Software« are bundled with the respective license text. The scope of this condition, in particular what is a substantial portion of the software and what is

A sound basic rule for users is to take the license text as literally and as seriously as possible.

45 ↗ see Duden. Die deutsche Rechtschreibung, 25., völlig neu bearbeitete Aufl.; ed. v. d. Dudenredaktion, Mannheim, Wien u. Zürich, 2009 (Duden Vol. 1), p. 317 u. p. 635
46 ↗ see Duden. Die deutsche Rechtschreibung, 25., völlig neu bearbeitete Aufl.; ed. v. d. Dudenredaktion, Mannheim, Wien u. Zürich, 2009 (Duden Vol. 1), p. 317 u. p. 635
47 cf. ↗ https://www.duden.de/rechtschreibung/Compliance
48 cf. ↗ https://de.wikipedia.org/wiki/Compliance_(BWL)
49 cf.↗ https://de.wikipedia.org/wiki/Compliance_(Law)
50 cf. ↗ https://opensource.org/licenses/MIT. We quote here the license template as specified by the OSI. The MIT licensed software itself always uses an instantiated license, i.e. a license with a specified copyright line. And if the license subsequently requires the license text and the preceding copyright line to be attached to the software, then it is precisely this specified copyright line that is meant. It is therefore not sufficient to add the template to the bundle that is passed on. You must always search for the actual license text in the repository to fulfil the license.

not, is ultimately a matter of interpretation. However, this does not mean that you can ignore the license or misinterpret this scope. A sound basic rule for users is to take the license text as literally and as seriously as possible. However, this also means that no action is required if the MIT-licensed software is not distributed, i.e. only used for personal purposes on personal computers.

In this way, **typical features of an open source license** are defined:

- the attribution of the rights of use
- the formulation of the conditions to be fulfilled
- the linking of the conditions to usage scenarios
- and the existence of room for interpretation that language always opens up.

In the following, we will take a look at further licenses, although not all details will be considered. After all, it is generally true that the appropriate fulfillment of the license conditions always requires a personal look at the license text. Secondary explanations – such as these – may help understanding, but they are irrelevant for the legal assessment, even if they claim to be a guide.

> The right to use Open Source Software cannot in fact be acquired directly with money.

Attentive readers who have already paid for Open Source Software will have wondered where payment is mentioned in such licenses. There are obviously no license fees. In fact, there is no contradiction here: the right to use Open Source Software cannot in fact be acquired directly with money.[51] Instead, you »acquire« the rights of use by doing what the licenses require. The minimum contribution that is required when Open Source Software is passed on unmodified is that users are correctly informed about the origin and license of this software. In other words: Open Source Software follows the principle of *paying by doing*.[52] So anyone who has nevertheless spent money on Open Source Software has paid for a service that is provided with and for Open Source Software[53], but not for the right to use the software for any purpose, to examine it, to pass it on to others and to modify it.[54]

This sets out the framework that needs to be filled when talking about open source compliance, namely

- about ↗ fundamental challenges
- about ↗ tools for compliance
- about ↗ marginal aspects of compliance and
- about ↗ basic legal constructs.

51 see the open source definition ↗ ch. 2.2
52 see: Reincke and Sharpe: Open Source License Compendium, Release 1.0.2, 2018, p. 103 ↗ https://telekom.github. io/oslic/releases/oslic.pdf
53 cf. the open source business models ↗ ch. 5
54 cf. ↗ https://www.gnu.org/philosophy/free-sw.html#four-freedoms

7.1 Open source compliance as a task

Software is generally subject to copyright[55]: By law, every software developer initially has the copyright to what he programs. As the author, he often assigns the majority of these rights[56], namely the so- called exploitation rights, to his company or customers by means of an employment or supply contract. This does not significantly change the initial situation: software becomes Open Source Software when the owner of the exploitation rights – be it the programmer or the downstream user – publishes the software under an open source license[57]. A license in turn becomes an open source license when the Open Source Initiative[58] – often abbreviated as OSI – includes the license in the list of official open source licenses[59]. And for this in turn, the license must meet the 10 criteria of the *Open Source Definition* (OSD)[60], which the *OSI* requires open source licenses to meet. It's as simple as that – in principle.

The benefit that *OSI* brings to the open source game with the *OSD* is that it uses the OSD to classify only those licenses as open source licenses that actually grant the corresponding rights to the users of the software. Anyone using software that has been published under a confirmed open source license can therefore be sure that it actually grants users the rights they need to use it. It is no longer necessary to carry out time-consuming legal checks before using software to determine whether this use is actually permitted. This makes it easy to use Open Source Software[61].

> Anyone using software that has been published under a confirmed open source license can therefore be sure that users are actually granted the rights they need to use it.

55 As a first approach to this topic, please refer to the two Wikipedia articles ↗ Software law and Copyright.

56 ↗ The right to be named as author cannot be assigned. In other words: still no one other than the programmer himself may call himself the author of the code.

57 In essence, there is a public list of official open source licenses managed by the Open Source Initiative: ↗ https://opensource.org/licenses/alphabetical

58 see ↗ https://opensource.org/about

59 In order to legitimately include a license, the OSI has given itself an *Approval Process*. ↗ https://opensource.org/approval

60 cf. ↗ https://opensource.org/osd The Open Source Definition is therefore only indirectly a definition of Open Source Software. In fact, it contains criteria that the open source licenses must meet, not the software. But these criteria refer to rights that the licenses must grant to the software users if they want to be open source licenses. Conversely, this means that users of software distributed under an official open source license always know that they have certain rights. However, nothing is said about their obligations!

61 ↗ From a formal point of view, it unfortunately remains the task of each user to make sure of this transfer of rights personally. Because only what has been decided in court is legally clarified. And whether the OSD is correct has, as far as we know, not yet been an issue in court. However, the fact that so little has been heard in court is also a sign of how little controversy there is on the subject. In other words: Anyone who adopts the OSD in matters of rights attribution and simplifies their life with it is in good company.

Nevertheless, open source compliance is a complex matter. This is because the license-compliant use of Open Source Software by actively fulfilling the corresponding open source license conditions must take various dimensions into account:

- The OSI's open source definition only specifies which rights the licenses must grant to users of software licensed in this way. The fact that these rights of use must be passed on free of charge results from the obligation to make the source code available at no more cost than can reasonably be charged for an (electronic) copy of the software and the rights to edit and pass on this code[62]. In addition, however, the *OSD* and the *OSI* leave it to the individual licenses to impose obligations on the users of the software – if and insofar as these obligations do not limit the rights to be granted. The rights to be granted are standardized, whereas the obligations are not. Therefore, it must always be determined on the basis of the individual license exactly which obligations must be fulfilled. In other words: What the *paying-by-doing principle* imposes on users in individual cases.

- Not all licenses that look like or claim to be open source licenses have been confirmed as such by the OSI. Some of them nevertheless meet all the OSD criteria and may still be accepted at a later date. Others are almost identical in wording to an already accepted open source license and yet will never be officially recognized as open source licenses due to a more or less minor deviation.[63] In addition, there are licenses whose »open source- ness« remains disputed.[64] And finally, one encounters licenses that claim to be open source licenses but in terms of content amount to a proprietary license. This is why a distinction is made between *genuine open source licenses, potential open source licenses, non-genuine or disputed open source licenses and fake open source licenses*. Nevertheless, even with these »secondary forms«, it follows directly from copyright law that persons who use software licensed in this way must comply with the requirements specified in the license: It is the right of any author to attach whatever terms of use to his work. The fact that a license is not a (genuine) open source license does not therefore exempt you from complying with its requirements.

> It is the right of every author to link their work to whatever terms of use they wish.

62 cf. ↗ https://opensource.org/osd Criterion No. 2. Source Code, No. 1. Free Redistribution, No. 3. Derived Works

63 So the JSON license is right, ↗ https://www.json.org/license.html

64 Such a »disputed« license is, for example, the *Server Side Public License* (SSPL), under which MongoDB was last relicensed. Initially, MongoDB was distributed under the AGPL-3.0, a recognized open source license (see ↗ https://opensource.org/licenses/AGPL-3.0). However, the company behind MongoDB felt that more and more MongoDB-as-a-service offerings were emerging in the cloud world without really taking the AGPL into account. At first, it wanted to clarify the meaning of the AGPL through a clause. Then it created its own license, the SSPL, which is identical in wording to the AGPL with the exception of one paragraph. The approval process at the OSI turned out to be »complex«, so that the company finally withdrew its application. (cf. ↗ https://en.wikipedia.org/wiki/Server_Side_Public_License). Nevertheless, the SSPL is still similar to the AGPL – even in terms of text. And from the point of view of copyright law, it does not matter whether a license is a real, a potential, a fake or a fake open source license: If one uses the software so licensed, it has to fulfill its conditions.

- Although the licenses can be easily classified by the nature of what they impose on the users of software licensed in this way as a requirement or prohibition – the following section is devoted to this – this can only be summarized at a more abstract level. Ultimately, the requirements of the licenses differ in detail in such a way that each user must determine for each individual case what exactly he must do (and refrain from doing). Classifications and groupings, as formulated below, are helpful. Taking them as the sole yardstick and ignoring the actual license text is inappropriate.

> An adequate open source compliance analysis therefore always includes an assessment of the deployment scenario.

- A further complication arises from the fact that the effect of open source licenses depends on the context: When which of its conditions must be fulfilled depends, among other things, on the type of use that triggers the requirement for fulfillment in the first place. One widespread trigger, for example, is the »redistribution« of Open Source Software, although the exact (legal) understanding of what constitutes redistribution in the legal sense can vary from country to country. Other triggers are the possibility of accessing the software via the network or the fact that the software has been modified. An adequate open source compliance analysis therefore always includes an assessment of the deployment scenario and whether the conditions triggered by it can be reconciled with the business purpose of the deployment.

- A final increase in complexity arises from the fact that the requirements for license-compliant use of Open Source Software also depend on the location in a software architecture where the open source component in question is to be used: Embedding a module or a library in your own work can trigger a strong copyleft effect; using Open Source Software as an independent component with its own address space is more likely not to. Attempts to manage such complex relationships with seductively simplistic rules – such as the requirement that Open Source Software with a (strong) copyleft effect should not be used at all – limit the scope of users excessively and put them at an unnecessary disadvantage compared to their competitors. It is better to also document the software architecture in the open source compliance analysis and to consider the context of use within this architecture.

The way in which software becomes Open Source Software is simple. The path to using it in compliance with licenses is sometimes rocky. For example, some managers declare open source compliance to be a mere risk that can be mitigated or simply adopted because you won't be »caught« anyway and – if you are – you won't be »punished« so severely in monetary terms. This view is wrong! You can only use Open Source Software legally if you comply with the license conditions – or not at all. Every company whose code-of-conduct obliges its employees to comply with the law also requires them to use Open Source Software only in compliance with the license – no matter how complex the conditions are.

> Legally, Open Source Software can only be used in compliance with the license terms – or not at all.

But there is also good news: in the end, things are becoming simple again. If you want to establish open source compliance for a product, you »only« need

- generate the corresponding software bill of materials (SBOM) by
  - creating a list of all open source components used in the product-specific software stack and
  - noting or linking to the homepage, version number, software repository and license for each entry and noting whether it is an app or program or a module or library,[65]
- document the software architecture,[66]
- have a list drawn up by compliance experts based on this information of what needs to be done to use the software in the product in compliance with the license[67] and
- work through this task list.

> From a systemic point of view, establishing open source compliance for a product becomes easy again in the end.

65 The work of gathering such information is that which is already best and most supported in terms of automation by open source tools. (cf. ↗ https://oss-compliance-tooling.org/ Tooling-Landscape/ OSS-Based-License-Compliance-Tools/)

66 A particularly good tool for documenting software architectures in diagram form is draw.io, hosted at https://diagrams.net/, published in source code at ↗ https://github.com/jgraph/drawio and published under the Apache 2.0 license ↗ https://github.com/jgraph/drawio/blob/dev/LICENSE

67 The tool-supported automated application of compliance competence is currently the least explored field. One approach at a very early stage is OSCake, ↗ https://github.com/Open-Source- Compliance/OSCake. This shows that we still have a lot of »manual work« ahead of us in open source compliance and the creation of predefined compliance artifacts – despite all our efforts.

7.2 License types and compliance activities

Before we offer you a classification of open source licenses and the conditions typically associated with them, we would like to explain the demarcation criterion used to form the categories. In fact, the usual classes

- *permissive open source licenses* without copyleft effect
- *Open source licenses with a weak copyleft effect*
- *Open source licenses with a strong copyleft effect*

depend on the way in which the *copyleft effect* is embedded in them. In this respect, it is also worth explaining separately:

»Copyleft« is a play on words: copyright serves to protect the rights of the author. Copyleft, on the other hand, aims to preserve rights once granted as common property.

7.2.1 The copyleft effect (as a demarcation criterion)

The term »copyleft« was invented by Richard Stallman as a »play on words«.[68] In his opinion, copyright owners use the copy**right** to deprive users of software of rights that should belong to them. In contrast, the copy**left** is intended to ensure that these rights cannot be taken away from users. Copy**left** therefore refers to a method »[...] to make a program (or other work) free and to demand that all modified and extended program versions are also free«. The author licenses his code in such a way that not only his own code may be freely used, freely modified and freely distributed, but also that all **modifications, additions and derivations** that other developers bring into the development process may be used **free of charge** in the same sense.[69] In short, »copyleft« means that you may only distribute your modifications under the same conditions under which you received the original version of your modifications.

In practice, it is also necessary to distinguish between **strong** and **weak copyleft**:

The *strong copyleft* wants to ensure that the software that uses a work licensed in this way as a constituent component is passed on under the same conditions under which one obtained the component embedded oneself. The original license terms thus

68 see ↗ https://de.wikipedia.org/wiki/Copyleft
69 see GNU project ↗ https://www.gnu.org/copyleft/copyleft.de.html

extend not only to modifications of or additions to the component itself, but also to the »custom« code that uses that component.[70]

The *weak copyleft* only wants to ensure this for the adopted work and its direct changes. This makes it possible to place independent code added to a library under weak copyleft as part of a derived work under a different license. Fewer or no restrictions apply to this other license. Only the embedded component distributed under a weak copyleft license is subject to these rules[71].

7.2.2 Open source license typology

This allows the known licenses to be classified into groups:[72]

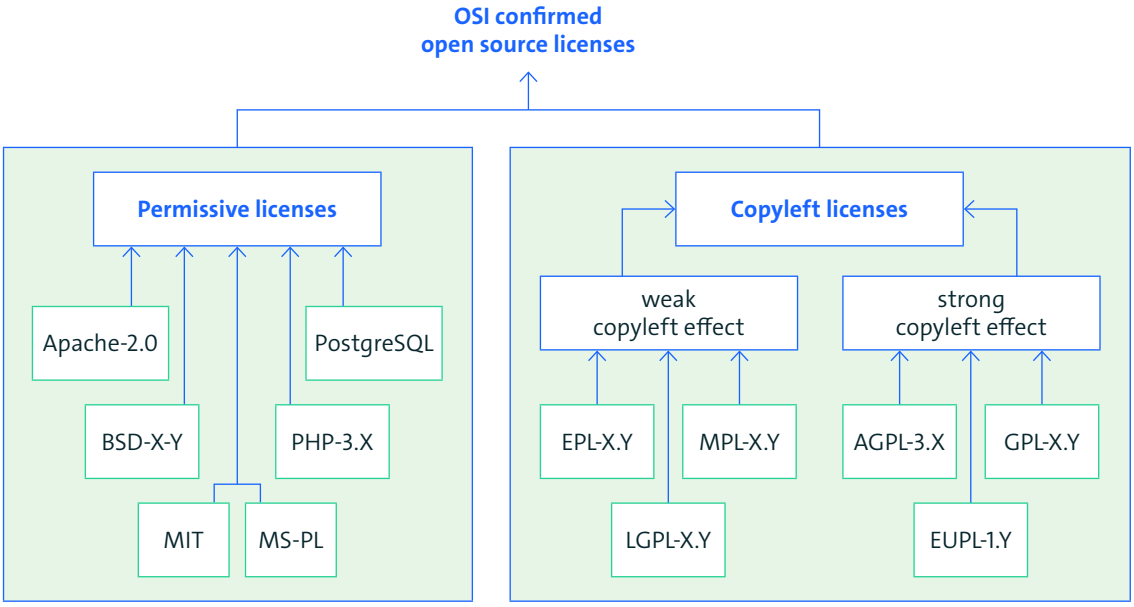

Fig. 1: Open source license classification (excerpt)

70 ↗ Thus, the *strong copyleft* can also cause license conflicts. If a program required two differently licensed libraries whose licenses each proclaimed a strong copyleft, then the program could no longer be distributed in compliance with the license – precisely because both embedded components demanded that the superordinate part be distributed under the same license as the component.

71 The OSI discusses abandoning the term *copyleft* altogether and replacing it with *reciprocal licensing*. The words weak and strong would then be replaced by the description of the »reciprocity scope«. (cf. ↗ https://opensource.org/node/875). We have already explained that strong and weak copyleft differ in terms of their effect and focus. It is impossible not to mention the term »copyleft«. There is hardly a more established term in the context of free Open Source Software.

72 see: Reincke and Sharpe: Open Source License Compendium, Release 1.0.2, 2018, p. 23 (↗ https://telekom.git- hub.io/oslic/releases/oslic.pdf). In addition to this, there is a second type of grouping in which the open source licenses are analyzed with regard to their handling of software patents. cf. this. op. cit. chapter 3.1 »The problem of implicitly releasing patents«.

Permissive licenses form the first subgroup of open source licenses. As OSI-certified licenses, they grant users all the rights specified in the OSD, in particular the right to use the software, to pass it on, to modify it and to pass it on in modified form (provided access to the source code is available.) In addition, permissive licenses **do not have a copyleft effect** – neither a weak nor a strong one. These licenses leave it up to the people working on the software to decide whether to redistribute their own work or their changes to the adopted work as free software. **It leaves them free to turn the result of their work into proprietary software** – hence the name of this category: *Permissive licenses* allow editors to keep the result of their improvements and changes to themselves, so to speak, even though that work is built on a free software base.

Permissive licenses also allow the editors to pass on the result of their interventions as proprietary software in binary form.

Nevertheless, every permissive license also imposes obligations on the users of the software licensed in this way, which they must fulfil even if they distribute their work as a whole in binary form under a different license.

- All permissive licenses require that packages or products containing such licensed software be accompanied by the license text.
- Some permissive licenses also expect special files to be included with these bundles if they are included in the repository
- Some permissive licenses also prohibit advertising with brands, names or similar.

The **group of permissive licenses** includes at least:

- *Apache-2.0*, the Apache license, version 2.0,[73]
- Berkeley Software Distribution Licenses,[74]
  - *BSD-2-Clause,*
  - *BSD-3-Clause,*
- *MIT*-License,
- *MS-PL* (Microsoft Public License),
- *PostgreSQL*-License,
- *PHP-3.0*-License.

> Permissive licenses bear this designation because they impose the fewest obligations on the user.

73 We follow the SPDX nomenclature (↗ https://spdx.dev/licenses/) for the license designation (the italic part), which may pursue the goal of simple and efficient retrievability of licenses through uniform designations and identifiers.

74 ↗ The MIT license and the BSD licenses require increased attention, as they are based on the idea of a template: One adopts the license text, specifies the copyright line and calls the whole thing XYZ license. In the case of BSD licenses and the MIT license, it is therefore necessary to look at the license text itself to determine whether it is really a BSD license or MIT license.

Licenses with a weak copyleft form the second subgroup of open source licenses. As *OSI*-certified licenses, they also grant users the *OSD* rights, i.e. in particular the right to use the software, pass it on, modify it and distribute it in modified form. The weak copyleft effect embedded in the license guarantees that users have access to the source code required for modification:

Namely, these licenses require – in addition to all the other conditions that they also want to see fulfilled – that changes and improvements to the software itself are used and distributed under the same conditions as those under which the processor obtained the original, i.e. under the same license. These conditions include that the software once obtained – modified or unmodified – must also be made available to third parties again in source code[75].

The open source licenses with weak copyleft do not, however, extend this obligation to the »higher-level« software, so to speak, which uses software licensed in this way with weak copyleft as an embedded component.

This means that licenses with a weak copyleft impose the following obligations on users:

- All licenses with a weak copyleft effect also require that packages or products containing software licensed in this way be accompanied by the license text.
- In addition, all licenses with a weak copyleft effect expect the source code of the licensed component to be made available in one way or another, regardless of whether the software has been modified or not.
- Some of the licenses with a weak copyleft effect also expect special files to be included with these bundles if they are included in the repository.
- Others want the copyright information of all the programmers involved to be displayed conspicuously, i.e. not only visible within the source code[76].
- Finally, some of these licenses also prohibit advertising with trademarks, names or the like.

In the case of licenses with a weak copyleft, the copyleft effect only affects the licensed software itself.

75 see also ↗ Copyleft effect in general
76 see ↗ https://opensource.org/licenses/LGPL-2.1, § 1

The group of licenses with a weak copyleft includes at least:

- *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).

Licenses with a strong copyleft form the third group of open source licenses. As *OSI* certified licenses, they also grant users the *OSD* rights, i.e. in particular the right to use the software, pass it on, modify it and pass it on in modified form. The strong copyleft effect embedded in the license guarantees that the user has access to the source code required for modification:

These licenses require – in addition to all the other conditions they also want to see met – that modifications and improvements be used and distributed under the same conditions as those under which the modifiers obtained the original, namely the modifications and improvements to the original software per se and those to the overarching »system software« that uses the original as a subcomponent. These conditions include that the modifications of the received software are made available in the source code as well as the new whole based on the original or the modified version[78].

In the case of licenses with a strong copyleft, the software that uses libraries/modules licensed in this way must also be distributed under the same license under which the libraries/modules used were distributed.

- All licenses with a strong copyleft effect also require that the license text be included in packages or products containing software licensed in this way.
- Furthermore, all licenses with a strong copyleft effect expect the source code of the licensed component itself to be made available in one way or another, regardless of whether the software has been modified or not.
- In addition, all licenses with a strong copyleft effect expect that the code of the software that uses the actual software as a component (in the same address space = library, modules) is also made accessible under the same license. This applies in particular to software that you develop yourself on the basis of these components: you are then no longer free to choose your own license.
- Some of the licenses with a strong copyleft effect also expect special files to be attached to these bundles if they are included in the repository.
- Others require that the copyright information of all programmers involved be displayed conspicuously, i.e. not only within the source code[79].
- Finally, some of these licenses also prohibit advertising with trademarks, names or similar.

77 ↗ The classification of the EPL is controversial, in some cases it is seen as a license with a strong copyleft.
78 see also ↗ copyleft effect in general
79 see ↗ https://opensource.org/licenses/GPL-2.0, § 1

The group of licenses with a strong copyleft includes at least

- *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 obligations in the overview

The subtleties of the licenses must not distract from one point:

The *paying-by-doing* principle applies to all of them. None of them offer the option of acquiring the rights of use financially while waiving the obligations. Instead, even the most »permissive« license expects users of such licensed software to do something under certain circumstances. In order to use Open Source Software in a license- compliant manner, it is not enough to think in generalized categories. Only the specific license determines what exactly is to be done under which circumstances. In the end, it's all about the individual case, about what the specific open source license actually requires. Nevertheless, it is sometimes helpful to take an overview – even if it does not claim to show all the details – as an indication of the direction from which one can expect requirements:

Class	License Text	Extra Documents	Component Code	System Code
Permissive licenses	must be supplied	must be supplied on a case-by- case basis	can be made accessible, but need not be	can be made accessible, but need not be
Licenses with weak copyleft effect	must be supplied	must be supplied on a case-by- case basis	must be made accessible	can be made accessible, but does not have to be
Licenses with a strong copyleft effect**	must be supplied	must be supplied on a case-by- case basis	must be made accessible	must be made accessible

80 The EUPL is a special case in several respects. For example, it is published in several languages, including German ↗ https://eupl. eu/1.2/en/. It also contains an explicit »Copyleft clause« (see § 5) and is generally classified as a strong copyleft (see ↗ https://www.ifross.org/artikel/public-license-eupl-22- sprachfassungen-verfuegbar). However, it also contains a »compatibility clause« which states that in the event of a »conflict«, the obligations of the other license »take precedence«, provided that this other license is mentioned in the list of »compatible licenses«. And the »EPL« also appears in this list – a license with a weak copyleft. (see also: ↗ https://telekom.github.io/oslic/releases/oslic.pdf p. 33 f)

This can be translated into rules of thumb, namely

- **for the permissive licenses**:
  - Distribute the license text together with the package (product) containing the software. The license text must be the one that really belongs to the software, not a license template. And actually distribute it together with the software, not secondarily via a download link.
  - Also distribute the files together with the package whose distribution is required by the license.
  - Refrain from doing what the licenses prohibit you from doing.

- **for licenses with a weak copyleft effect:**
  - Distribute the license text together with the package (product) containing the software. The license text must be the one that really belongs to the software, not a license template. And actually distribute it together with the software, not secondarily via a download link.
  - Also distribute the files together with the package whose distribution is required by the license.
  - Make the source code of the component used by your system accessible. A reference to existing repositories is not sufficient. And make the code of the exact version that is installed in your system accessible.
  - Refrain from doing what the licenses prohibit you from doing.

- **for the licenses with a strong copyleft effect**:
  - Distribute the license text together with the package (product) containing the software. The license text must be the one that really belongs to the software, not a license template. And actually distribute it together with the software, not secondarily via a download link.
  - Also distribute the files together with the package whose distribution is required by the license.
  - Make the source code of the component used by your system accessible yourself. A reference to existing repositories is not sufficient. And make the code of the exact version that is installed in your system accessible.
  - Finally, make the source code of the higher-level system you have developed that uses the component with a strong copyleft accessible. A reference to existing repositories provided by you is not sufficient. And make the code of exactly the version that is built into your system accessible here too.
  - Refrain from doing what the licenses prohibit you from doing.

- **for all licenses:**
  - Carefully observe the license-specific *Dos and Don'ts*
  - Do not base your compliance analysis on instructions that inappropriately condense the subject matter – as these rules of thumb do.

7.3 Open source compliance tools

We can see that a lot needs to be done to make an open source-based product license compliant. And this will have to be done under complex conditions. Ultimately, it always comes down to aggregating and generating the right compliance artifacts in the right context and integrating them into the package/product in such a way that customers can view them. This task calls for thematic and technical support.

7.3.1 Training

A central German-language source on *open source compliance* is the book »Open-Source-Software. The legal framework of free software«. It explains legal contexts and historical backgrounds[81].

Another tool is »ifrOSS«, the »Institute for Legal Issues of Free and Open Source Software«[82], which claims to have set itself the task of »[...] accompanying the rapid development of free software from a legal perspective«[83]. If the advantage of the book is its systematic condensation, then the advantage of this site, which is accessible on the Internet, is its diversity and topicality.

As a third tool, the OpenChain project offers extensive comprehensive training material.[84] Material on open source licences can be found on the pages of the OSI, the *Open Source Initiative*[85] and for specific licences it is advisable to refer to the pages of the organizations that maintain the respective licence, if such an organization exists.

81 see Jaeger, Till and Axel Metzger: Open-Source-Software. Rechtliche Rahmenbedingungen der Freien Software; 3rd ed: C. H. Beck 2011. This book is currently in its 5th edition and will – it is to be hoped – be further updated. Other inspiring sources could be, for example, the handbook »Open Source for Business« [see Heather/Meeker, »Open Source for Business – A Practical Guide«, 3rd edition 2020], the commentaries on the important open source license GPL in the copyright commentary [see Fromm/Nordemann/Czychowski, Urheberrecht, 12th edition 2018, GPL, from page 2779] or books on copyright law in general [see for example: Wandtke/Bullinger/Grützmacher, UrhR, 5. Edition 2019, Section 69c para. 73 et seq.

82 cf. ↗ https://www.ifross.org/

83 see ifrOSS: Objectives, tasks, history, ↗ https://www.ifross.org/? q=node/16

84 see ↗ https://www.openchainproject.org/resources

85 see ↗ https://opensource.org/

7.3.2 Advice

Users who want to integrate Open Source Software into their product now have a number of people and institutions on hand to help them achieve open source compliance. This does not just start with well- known, internationally recognized lawyers, but extends to established companies that offer a full service and finally leads to the Open Chain Project[86], which is dedicated to the topic of global open source compliance across supply chains. Describing this in more detail is of particular value due to its overarching role:

As a project, the aforementioned *OpenChain* is affiliated with the *Linux Foundation*.[87] It forms, at its core, »a diverse, global commercial partner network« through which to ensure that any organization of any size can apply the industry standard that OpenChain is developing.[88] While this standard initially consisted of the conception of an ideal structure against which companies could self-certify,[89] OpenChain* has recently succeeded in having its standard officially elevated to the »International Standard for Open Source License Compliance« with the number *ISO 5230*.

The basic idea behind *OpenChain* is therefore that the more suppliers that adhere to this standard, the easier it will be to use open source- based supplier products. It goes without saying that there is still a long way to go before the uncontrolled adoption of preliminary products and the corresponding compliance artifacts.

In addition to its central tasks, *OpenChain* manages a number of »work groups«, including the »sub-project« *Open-Source-Tooling for Open-Source-Compliance*, which claims to be »[...] focused on reducing reSource costs and improving the quality of results around open source compliance activities.«[90] In fact, the focus here is on linking existing open source tools to form a closed »tool chain« that automatically generates the compliance artifacts that must be included with your open source-based product. To this end, this group also maintains an excellent overview of programs that support the generation of compliance artefacts[91].

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

7.3.3 Tools

If you remember what needs to be done to prepare your product in a license-compliant manner, it is easy to see that the work to be done can be typified:[92]

- The **Analyzer** determines the dependencies: You need to know which Open Source Software belongs to a product as a whole.
- The **Downloader** saves the content of the corresponding repositories on the local system so that it can be viewed with regard to the licenses.
- The **Scanner** then determines the actual licenses and marks elements that can be incorporated or converted into compliance artefacts.
- The **Evaluator** allows you to introduce your own usage rules into the process and have them applied.
- The **reporters** aggregate the information into the required compliance artifacts.

Depending on the intended purpose, different reporters can be defined and integrated.

If you want to understand which tools are ready for use and can be used in this task suite, take a look at the *OpenChain-ToolingLandscape*[93]. And if you want to see to what extent these can actually be linked to chains, take a look at ORT[94] or QMSTR[95].

However, there is currently no continuous chain. It is therefore not yet possible to fully automate the creation of compliance. The good news is that the open source community is already working on this.

> It is therefore not yet possible to fully automate the creation of compliance.

One-fits-all approach. What is missing is a component that brings legal licensing knowledge to the table and applies it to each individual case so that suitable artefacts are created depending on the context. OSCake, the *open source compliance artifact knowledge engine*, is working on this desideratum. (see ↗ https://github.com/Open-Source-Compliance/ OSCake). Once this has been implemented, compliance will be truly automated.

92 We follow here – slightly modified – the terminology from ORT, the *Open Source Revie*w Toolkit. It is a »meta-system« that uses existing open source solutions for the individual compliance tasks as far as possible. (see ↗ https://github.com/oss-review-toolkit/ort). As a superordinate tool for generating compliance artifacts, ORT already goes a long way, especially as it can be configured very flexibly. However, it still works with a One-fits-all-approach. But a component that enables the creation of suitable artefacts by using legal know-how depending on the respective matter is still missing. A solution to this problem is being worked on with the Open- Source-Compliance artifact knowledge engine, called OSCake. As soon as a solution is found, compliance will be completely automated.

93 cf. ↗ https://oss-compliance-tooling.org/Tooling-Landscape/OSS- Based-License-Compliance-Tools/

94 cf. ↗ https://github.com/oss-review-toolkit/ort

95 see ↗ https://qmstr.org/

7.4 Special challenges

Many open source licenses date back to the last century or the early 2000s[96]. There was no »digital biotope« back then. Computers were used differently than they are today: if you wanted to use a program, you had to get a copy that matched your own operating system. Entire distributions were sold on floppy disks or CDs with program collections were enclosed with computer magazines.

So even today – despite the new ways of use – the distribution of software still triggers compliance with open source license obligations. In terms of language, the current licenses are still aimed at what was current when they were created. This occasionally causes a discrepancy between modern technology and traditional licenses. Nevertheless, we are also required to comply with these »old« licenses. Ignoring them is not an option.

The only way out of this dilemma is to »think along« with the special challenges described below.

> The fact that software is passed on is still the main trigger for the activation of license requirements to be complied with.

7.4.1 SPDX and license naming

The abundance of different licenses and their variants sometimes makes it difficult to know which license is involved. This problem has been addressed by *SPDX* and the names have been standardized[97]. This approach has become so well established that the *SPDX identifiers* are now also used in the source code and thus become part of the *Licensing Statement*, unless added later.

7.4.2 The Javascript challenge

Javascript is considered a »lightweight, interpreted or JIT-translated language.«[98] Such programs are often – but not only – sent to computers as part of a web front-end, where they are executed by a Javascript interpreter built into the browser. Many

96 The BSD licenses were used for the first time in the early 1980s (cf. ↗ https://www.lin-fo.org/bsdlicense.html); the MIT license dates from 1988 (cf. ↗ https://de.wikipedia.org/wiki/MIT-Lizenz), but also has an eventful history (cf. ↗ https://opensource.com/article/19/4/history-mit-license). GPL-2.0 and LGPL-2.0 date back to 1991 (cf. ↗ https://opensource.org/licenses/GPL-2.0 and https://opensource.org/ licenses/LGPL-2.0), LGPL-2.1 to 1999 (cf. ↗ https://opensource.org/licenses/LGPL-2.1), Apache-1.1 to 2000 (cf. ↗ https://www.apache.org/licenses/LICENSE-1.1), Apache-2.0 to 2004 (cf. ↗ https://www.apache. org/licenses/LICENSE-2.0) and the (L)GPL-3.0 licenses to 2007 (cf. ↗ https://opensource.org/licenses/GPL-3.0 and https://opensource.org/licenses/LGPL-3.0))

97 cf. ↗ https://spdx.dev/licenses

98 see ↗ https://developer.mozilla.org/de/docs/Web/JavaScript

Javascript libraries are MIT licensed – simply because the source code is always implicitly supplied anyway. According to this license, the license text must be »included« in this library and distributed with it[99].

To save bandwidth, however, the standard of »minification« has emerged, which removes all whitespace and all comments from the source code.[100] This makes the discrepancy apparent: compliance with the license requires something that the technology does not reasonably implement.

How can people focused on compliance deal with this? They could, for example, use the non-minimized version. This would slow down the network in the long term. Or they could reintroduce the license text in their versions before delivery. This would require complex integration into their own CI/CD pipeline. Or they could make the license texts available for download themselves. This would mean that the code and license would not be sent as a single unit. This creates a tragic situation in the classic sense: whatever you do, it's wrong.

Fortunately, the situation eases insofar as the developers of the libraries themselves offer them as minimized versions and thus seem to implicitly approve of the procedure. In addition, most »minifiers« now support the preservation of comments relevant to the license.

> Javascript is – technically speaking – always open source. But the code is not necessarily Open Source Software. This is decided by the licensing.

7.4.3 The AGPL or network usage as a compliance trigger

> The AGPL is intended to guarantee free software even in times of cloud technology.

As mentioned, the open source licenses have a certain age and are technically linked to the status that was current when they were created. This is why the transfer of software usually triggers compliance rules. In times of web-based services, where data rather than programs are exchanged, fewer compliance cases are triggered: The Open Source Software is operated as a service in the cloud or in the data center and is simply not passed on. This is formally correct, but does not quite correspond to the idea of free software.

AGPL-3.0 was developed to close this gap. It is similar to GPL-3.0 with a single exception: the added § 13 stipulates that all obligations of the license must also be fulfilled towards users who interact with the program »remotely through a computer network«[101].

99 The MIT license explicitly states »The above copyright notice and this permission notice **shall be included in all copies or substantial portions of the Software**.« cf. ↗ https://opensource.org/licenses/MIT [orig. KR.]

100 cf. ↗ https://www.cloudflare.com/de-de/learning/performance/why- minify-javascript-code/

101 cf. ↗ https://opensource.org/licenses/AGPL-3.0, § 13. However, the situation is somewhat more complicated: On the one hand, the AGPL limits the trigger here to cases where the service provider has modified the program. In addition, however, it then requires him – in a linguistically rather imprecise manner – that all additional parts must also be distributed under the same license. Unwilling interpreters might understand this to mean the entire software stack.

Anyone offering services on the Internet should therefore still be vigilant. It is not enough to derive license compliance from the fact that the software is not distributed. Rather, all AGPL-licensed components must be analyzed separately.

7.4.4 (L)GPL-v3 and replaceability

The 3rd generation of (L)GPL licenses also contains the central component of all GNU licenses, the copyleft effect. At the time of the revision of the 2nd generation[102] – 2005 to 2007 – there were two other pressing issues in the open source community, namely the handling of software patents[103] and »digital rights management«.

With regard to DRM, GPL-3.0 requires, on the one hand,[104] that anyone who uses software licensed in this way in systems »waives his right to prohibit circumvention of technological protection measures«.[105] On the other hand, GPL-3. 0,[106] »[...] that when distributing devices with GPL software, the necessary information for installation [...] must be supplied in order to be able to re-install edited GPLv3 programs«.[107]

(L)GPL-3.0 licensed software requires special care if it is to be integrated into »embedded systems«.

Both stipulations together have a dramatic impact on the use of (L)GPL-3.0 licensed software in devices: If it is used, its manufacturer must make it technically and/or procedurally possible for everyone who receives such a device to compile and install improved versions on it. And they may not contractually restrict the use of this technical option. The manufacturer may not even prohibit its »customers« from »hacking« these devices in order to be able to install improved versions.

These requirements apply to Internet of Things systems with (L)GPL 3.0 software as well as cars, trains, washing machines and many more.

Product managers must therefore carefully consider how (L)GPL 3.0 licensed software should be handled in the closed system. They could generally ban it from products that they do not want to give an interface to replace it. Alternatively, they could have their device designed with such an interface. Finally, they could also pursue the idea of having their customers' own developments imported by their own team. If they did this, they would be well advised to regulate the warranty separately and to provide for post-certification for legally approved devices. In the case of car tuning in Germany, this is a common procedure with the registration of approved modifications by the TÜV.

However, in »closed« systems it is important to keep a close eye on the (L)GPL-3.0 licensed software.

102 see Jaeger, Metzger: Open-Source-Software. Munich 2011, p. 50 ff.
103 cf. ↗ https://www.fsf.org/blogs/community/the-threat-of-software- patents-persists and https://opensource.org/licenses/GPL-3.0, § 11
104 cf. ↗ https://opensource.org/licenses/GPL-3.0, § 3
105 cf. Jaeger, Metzger: Open-Source-Software. Munich 2011, p. 61.
106 cf. ↗ https://opensource.org/licenses/GPL-3.0, § 6
107 cf. Jaeger, Metzger: Open-Source-Software. Munich 2011, p. 61.

7.4.5 Open source compliance, automatic updates and CI/CD chains

Today, it is common to install firmware updates via the network. Such a newly assembled stack must of course be reviewed again with regard to open source compliance. This often requires new compliance artifacts to be generated.

This is because licenses may change from one version to the next, copyright lines may have been added and files to be included may have been adapted.

This poses a particular challenge for product maintenance: anyone who maintains a CI/CD pipeline to generate and install updates for their products should integrate the generation of compliance artifacts into it. And anyone who generates new versions »manually« will also want to repeat the compliance preparation »manually«.

Anyone who maintains a CI/CD pipeline to generate and install updates for their products should integrate the generation of compliance artifacts into it.

7.4.6 Maven or automatic package aggregation

Maven[108] is the prototype for a special compliance challenge: the basic idea of this tool is that the developers define the components of their project in a POM file[109] and that *Maven* executes the build process independently and reproducibly on this basis.[110] For this purpose, the open source components required in each case are brought in 'on the fly' from an (external) repository into the local development environment.[111]

In the POM, the components may also be defined 'underspecified': If the release number is missing, *mvn* fetches any version. If the location is missing, *mvn* fetches the components from somewhere and so on. This simplifies the development work and weighs down the compliance supervision. For the time being, it remains systemically unclear which software has really been built into a product. Even if the POM file has been formulated in detail, network irritations can still cause maven to 'exceptionally' use another repository. This means that the content of the referenced packages – also with regard to supplied compliance artefacts – no longer has to be the same. If open source compliance is worked out on a certain *maven* status, the next call to *mvn build* may have already changed the basis of the compliance work.

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

For this problem, there are at least approaches and ideas for solutions:

For example, one could radically distribute their program as »in-house development + manifesto«. Guaranteed reproducibility should ensure that the build process also runs on other systems. The developer would not need to generate compliance artifacts for the components that are collected on the customer's computer. In a sense, it is not the developer who gives the software to the customer, but *mvn*. The developer would therefore only be responsible for the license- compliant handover of their own development. However, this approach has at least the disadvantage of a very difficult warranty: after all, how can a company be held liable for something that is finally assembled on the customer's computer? If something doesn't work, customers will turn to their supplier again, even if the cause of the »fail« lies on the customer's computer. And the producer has at least the effort to determine and prove this.

Whenever a developer has their software packages put together transparently, the open source compliance work needs to be specially designed.

Alternatively, the developer could move the components to their own »company« repository and have the manifests referenced last. This would give them control of the inventory and a solid basis for their compliance activities, but they would have to accept increased effort in terms of repository and version maintenance.

Maven is not the only automatic aggregation system. The same applies to other approaches: whenever a developer has their software packages compiled transparently, the open source compliance work must be specially designed. Because ignoring the license requirements is not an option.

7.4.7 Compliance in the Cloud: Virtual Machines

Clouds that operate with virtual machines also have their own special compliance challenges. Their architecture looks like this – with the proviso that it is adapted:

- On the lowest level are the actual computers, which may be running an open source operating system such as *Linux*.
- In the middle, this machine combines a virtualisation layer – such as Openstack[112] – to form the actual cloud. And this layer can also use Open Source Software: Openstack, for example, is distributed under Apache 2.0.
- Finally, it is part of such systems that the virtualisation layer not only provides the requested virtual machines with memory and disk space, but also with a bootable software image. Without this, it would not be able to start the virtual machine.

112 see ↗ https://www.openstack.org/

From the perspective of a compliance officer, the situation looks like this:

- It is the responsibility of the hardware suppliers to ensure that the computers themselves are handed over to the cloud operators in a license-compliant manner.
- It is the responsibility of those who install the virtualization layer on the hardware to ensure that it is handed over to the cloud operators in a license-compliant manner.
- Software is handed over to the users of a virtual machine with the images, at least if this image includes ssh access. This is because anyone with ssh access will usually also have scp access and can use it to download the software on their virtual machine.

This means that in a cloud that operates with virtual machines, the person who provides the image must ensure that it is prepared in accordance with the license. This can be the person who orders a virtual machine – provided they bring their own image and have it integrated into the Openstack retrieval process. In most cases, however, the images are provided by the cloud operator as part of the virtualization layer. This means that the cloud operator is responsible to its customers for generating and providing the compliance artifacts for all open source components used in the images.

7.4.8 The strong copyleft without a strong copyleft

Libraries that are distributed under a license with a strong copyleft force the works based on them to be distributed under the same license. So far, so familiar, so uncritical.

Exceptions are another method of limiting the strong copyleft effect.

It becomes strange when a central core library at the lowest level of a software stack is licensed in this way. The developers of the GNU system have also considered this effect and have therefore placed the glibc, i.e. the library that enables calls into the kernel and is used by every other component,[113] under the LGPL,[114] not under the GPL.

OpenJDK[115] is different: This free Java standard library is licensed at its core under the GPL-2.0. And like libc, it is used by all Java programs, provided it is installed as a Java standard library. This could indicate a clash. In fact, OpenJDK is published under the *GPL-2.0- with-classpath-exception*: Although this confirms the generally strong coypleft effect, it grants an exception for this library[116] and reduces the effect to a weak copyleft.

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

When considering compliance, it is important to bear in mind that not every license with a strong copyleft triggers the strong copyleft effect under all circumstances.

7.4.9 Upstream compliance

If you publish something as open source – the term »giving something upstream« has now become commonplace – there are two variants:

Firstly, you can set up your own project with its own repository. In this case, the freedom to choose a license is only limited by the strong copyleft effect of required components. Irrespective of this, publication should be accompanied by other users fulfilling the conditions of our license. It is therefore advisable to include the artifacts required by the selected license in the repository. This creates clarity.

On the other hand, you may want to contribute to an existing project. This will only be successful if you submit to the customs of the project. This goes as far as licensing and signing a possibly required contributor license agreement. For both, REUSE[117] is a new procedure that standardizes the filing of licenses and licensing statements.

7.4.10 Export control

Modern software development operates across borders; software is delivered across national borders. With regard to the open source phenomenon, this often involves redistribution (redistribution) when components of software are previously downloaded from the Internet and distributed in the course of product sales.

However, this usual procurement of software can lead to components being included in the product that are neither approved for use in Germany nor may be delivered from there to certain countries outside Germany. International companies in particular are required to take this aspect into account when setting up their supply chain. There are a number of references to this on the Internet[118].

117 cf. ↗ https://reuse.software/

118 see for example ↗ https://www.apache.org/foundation/license- faq.html or https://www.cloudfoundry.org/ and information on embargo options.

7.4.11 Compliance and software patents

Patents serve to protect technical inventions and grant patent holders a temporary monopoly position in return for which they must disclose the technical teaching.

In the area of software, however, legislators have opted primarily for protection under copyright law. Thus, computer programs are explicitly protectable under copyright law as linguistic works,[119] whereas in patent law, programs for data processing systems are initially expressly not regarded as inventions.[120] However, this only clarifies that an implementation as software is not in itself regarded as an invention. The decisive factor for patentability is whether there is a technical reference, so-called technicality. For example, claims that involve the execution of certain process steps by a computer to solve a problem in the classical fields of technology are already patentable. In addition, a protectable, computer-implemented invention may exist if it has special features that justify patentability. This is often controversial in individual cases. However, patent protection for pure application programs is usually rejected.

Since software is protected by copyright, open source licenses also operate primarily under copyright law.

Since software is protected by copyright, open source licenses also operate primarily under copyright law. Due to a possible parallel protection under patent law, side effects may arise under certain circumstances. For example, in the case of purely copyright-based open source licensing, the rights of use could be limited by patents where action is taken against the copyright licensee on the basis of existing patents.

Patent clauses in open source licenses serve to resolve this conflict. There are two approaches:

- A patent license is granted in addition to the rights of use under copyright law. Examples of open source licenses with a patent license are the
  - GPL-3.0 (and variants)
  - AGPL-3.0
  - Apache-2.0
  - MPL-1.1/2.0
  - EPL-1.0/CPL-1.0

- A patent retaliation clause is attached to the open source license which, although it does not grant any rights of use under patent law, cancels the rights of use granted under copyright law in the event of a patent action. Examples of open source license patent retaliation clauses are the
  - Apache-2.0
  - MPL-1.1/2.0
  - EPL-1.0/CPL-1

119 cf. German Copyright Act (Urhebergesetz) UrhG § 2 para. 1 and §§ 69a
120 cf. German Patent Act (Patentgesetz) PatG § 1 para. 3 and 4 and correspondingly EPC Art. 52 para. 2 and 3

Consequently, patent clauses are now common provisions in open source licenses. In addition, there are licenses such as the MIT license that formulate the granting of rights very broadly and not in a copyright-specific manner, so that the granting of patent rights could also be included.

As a rule of thumb for license-compliant use, patent holders can assume that by passing on Open Source Software, they implicitly grant a right to use the patents in their patent pool that are necessary for the use of the software. However, this only applies in the context of this software and not in general.

7.5 The international legal basis

So far, we have explained what needs to be done in order to use Open Source Software in a license-compliant manner. Another interesting question is why these licenses have to be observed at all. On what basis are these obligations imposed on us when it comes to international issues? The following section examines this legal question.

The so-called »principle country country protection« is generally applied in the context of issues that are regularly characterized by copyright law. This basically means that the legal structure of copyright is based on the law of the respective country for whose territory protection is claimed. In the open source context, this applies to central legal issues such as who is actually considered the author and first rights holder of programming, how they can grant rights to their programming to third parties, whether their rights are transferable at all and how they can tailor these rights, i.e. what content the rights granted have.

For all contractual issues, however, the so-called statute of contract applies. In Europe, this is governed by the Rome I Regulation (Rome I Regulation). According to Art. 3 Rome I Regulation, any choice of law takes precedence. However, since many open source licenses of great practical importance (e.g. GPL, MIT, BSD and Apache licenses) do not contain a choice of law clause, the law of the author's or rights holder's domicile regularly applies. This is because if the parties do not explicitly choose a law, Art. 4 Rome I Regulation refers to the place where the characteristic service is provided.

According to the prevailing view, in the case of the granting of simple rights of use – as in all open source licenses – this lies with the licensor, i.e. the author.

According to Art. 12 of the Rome I Regulation, the scope of application of the contract statute includes questions of formation, validity, liability and warranty, as well as questions of interpretation. The latter is relevant, for example, for the interpretation of the scope of a copyleft effect of a license or the question of whether the license also grants rights to use software by way of a SaaS offer.

8 Outlook

Writing about the fact that Open Source Software has conquered its recognised place among providers of products, software and services in the information economy, telecommunications, new media and the digital economy, and that it is impossible to imagine current and future products and innovations without it, seems somewhat understated when measured against its outstanding success. The concept of Open Source Software is now 40 years old and has always been subject to challenges. Nevertheless, the ecosystem has become stronger and more powerful. While in the early years it was mainly enthusiasts and volunteers who created software in the private sphere, today the topic is ubiquitous in the professional environment as well. It is increasingly taken for granted in companies across functions such as software development, but also other areas such as purchasing, marketing and human resources. The majority of developers who contribute to Open Source Software do so as part of their paid work[121].

> The majority of developers who contribute to Open Source Software do so as part of their paid work.

Open Source Software has become »mainstream« and has arrived in every company and all business sectors.[122]

But of course there are still a number of challenges to be overcome in the future, for example through the rapidly growing model of operating Open Source Software as a service. However, if one no longer operates the software oneself, but consumes it as a service, then the freedoms guaranteed by open source licences may only have a limited effect on the users of the services. This can even lead to a strengthening of manufacturer dependencies. Nevertheless, it is possible to migrate data and use cases to another service provider or to operate the software oneself. It is crucial for users to build up the necessary competence to be able to act confidently.

Furthermore, for companies that offer open source-licensed products, there is a risk that their business model is no longer viable if it had not taken into account the possible use in the cloud context. For example, some companies have changed their licensing model due to the practices of the so-called hyper scalers and no longer rely on generally accepted open source licences, but on their own, unrecognised or even proprietary models, citing pressure from the competition of the large cloud providers as the reason. A possible and already practised response of the ecosystem to licence changes in such cases are forks of the respective open source product.

This problem is an indication that there is a need to further develop open source licences. Another indication is the emergence of so-called »ethical source licences«, which see themselves as a further development of open source licences, but exclude certain use cases that are perceived as ethically unacceptable. The challenge here will be to draw clear lines between what can be regulated in a software licence and what must be regulated through other mechanisms, and to maintain the spirit of the »open source definition«.[123] [124]

121 See the Linux Foundation's »FOSS 2020 Contributor Survey«. ↗ https://www.linuxfoundation.org/resources/publications/foss- contributor-survey-2020
122 Open Source Monitor 2021, ↗ https://www.bitkom.org/opensourcemonitor
123 ↗ http://wiseearthtechnology.com/blog/peaceful-open-Source-license, https://github.com/atErik/PeaceOSL
124 ↗ https://www.golem.de/news/open-Source-lizenzen-gegen-den- missbrauch-freier-software-1509-116210.html

In general, it can be stated that in the future it will be very important to distinguish where open source is merely used as a nice marketing term or as an ideological cudgel and where open source actually delivers on the promise of more innovation, efficiency, transparency and independence.

Other challenges will follow and the open source ecosystem will master these as well. Software development in and with the ecosystem is simply necessary for many companies today in order to continue to bring competitive software-based products and solutions to market in a timely manner. Active participation in the ecosystem can, if necessary, alleviate the continuously growing shortage of developers. Especially for companies from states facing demographic change, the »open source way« can be an option to mitigate the effects of the skills shortage.

Software development in and with the ecosystem is simply necessary for many companies today to continue to bring competitive software-based products and solutions to market in a timely manner.

Open Source Software has positively influenced innovative operating models. It has provided a creative, reliable and, above all, competitive extension of use both in corporate procurement processes and in classic tendering situations. In the coming years, the worldwide open source movement will have to continue to face the task of maintaining and, in many cases, expanding its lead as a serious alternative and competitor to proprietary software in the market. This requires a targeted adaptation and modernisation of its licensing landscape as well as the achievement of higher degrees of standardisation and the establishment of a comprehensible quality awareness. A decisive factor is to make the sometimes still time-consuming verification of compliance with all its aspects a matter of course in software asset management through structured automation. Standards such as OpenChain ↗ ISO/IEC 5230:2020 and ↗ SPDX will simplify licence compliance along the software supply chain in the coming years. But licence compliance will remain an exciting topic. Companies are well advised to build up the corresponding competences. The Open Source Programme Office (OSPO) has established itself as an effective instrument for integrating open source compliance into a holistic open source approach. All open source issues are bundled in it: From compliance with regard to use and own contributions to the support of open source topics in business development and the establishment of ecosystems for the establishment of innovative service architectures with partners and customers.

In the past, numerous reference implementations or Defakto standards were already realised as Open Source Software. In the future, too, disruptive solutions based on Open Source Software will prevail and establish a de facto standard if they strike a chord with users. However, it is to be expected that more and more often a reference implementation of a defined standard will be developed cooperatively on the basis of open source. It can thus be assumed that standards will be easier to use in the future and will thus become established more quickly.

Attacks on the software supply chain, in which malicious code is introduced directly or indirectly into supplied software, have increased in the recent past and this trend is set to increase massively. This affects all software (and hardware), regardless of which licence it is under. However, Open Source Software offers special challenges here, but also opportunities. For example, the large number of open source modules used in some language ecosystems can make it difficult to verify the integrity and trustworthiness of sources and packages used. On the other hand, the transparency of open source makes it possible to develop effective mechanisms to carry out checks oneself, where proprietary software leaves only trust in the manufacturer. It is advisable to exercise care in the integration of supplied software and to support community-wide initiatives to secure the open source supply chain. Due to the increasing spread of Open Source Software and the rising degree of dependency, the number and intensity of code reviews will increase and thus more security vulnerabilities will be found and fixed. Nevertheless, dealing professionally with the »perceived insecurity« is a challenge.

Open Source Software is a powerful driver of innovation in many ways. For example, innovative open source solutions can be used in areas where they were not previously present without any significant upfront investment. The availability of the source code, the right to change it, thus being able to adapt it to the requirements of the respective areas, and finally the royalty-free nature of open source make this possible. In this way, completely new products and even markets can emerge. In short: Open Source Software is an enabler for new technologies, it enables the opening up of new markets and the development of future business models. While the use of Open Source Software is now ubiquitous and has already become a matter of course, most companies still have the next step ahead of them: active participation in the open source ecosystem. Taking the next steps here, establishing a culture of sharing and contributing pragmatically and ultimately also strategically to open source projects opens up enormous potential – not only from an idealistic point of view, but above all from an economic one. The innovative power and development capacity of the open source ecosystem cannot be surpassed by any single company. Therefore, it is simply a must to actively participate in the ecosystem in order not to be sidelined technologically. Open source foundations (cf. ↗ Chapter 6.4) will continue to play a central role here, as the legal framework they provide provides a neutral environment in which companies can develop software together. This creates the conditions for competing companies to cooperate in non-competitive areas on the basis of open- source software.

> Open Source Software is an enabler for new technologies.

> 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.

»Bernard of Chartres said that we are, as it were, dwarfs sitting on the shoulders of giants, in order to be able to see more and more distant things than they do – not, of course, thanks to our own keen eyesight or height, but because the height of the giants lifts us up.« – John of Salisbury c. 1120: Metalogicon 3,4,47–50[125]

In this sense, we wish all companies and developers the courage to take the step towards openness, and thus to open up the full possibilities of open source for themselves and all of us.

125 ↗ https://de.wikipedia.org/wiki/Zwerge_auf_den_Schultern_von_Riesen

9 Excursus

9.1 On the emergence of Open Source Software

Open Source Software also has a history as an idea[126], knowing about it facilitates understanding – and this is the task the following sections are dedicated to.

9.1.1 From Unix to Linux

The exchange of improved versions of the software among customers was seen as an improvement of the hardware product.

The term »Open Source Software« itself is a fairly recent invention. It was only created in 1998 – as a »marketing gimmick«, so to speak – to replace the term »free software«, which had been in use since the early 1980s. The term »free software« also sent (and still sends) political signals. Thus, the desire for a pragmatic, harmless alternative arose in the community.

Apart from this programme, however, free software has been present **since the beginnings** of commercial computer technology: in the first decades, free software formed a constitutive component of computer companies' business model, without being called free software yet. The actual business of commercial computer science initially consisted »only« of selling hardware. Software was the »give-away« that was delivered together with the hardware, including the source code. The exchange of improved versions of the software among customers was seen as an improvement of the hardware product. This approach was favoured by the nature of the clientele: early computer systems were mainly based at universities. Scientists and technicians used the software supplied, adapted it – as part of their work – to their respective needs, passed it on, received modified versions, and based their own work on it.

In this way, they countered the lack of possible uses for the expensive hardware.

Conversely, it was in the interest of the companies that the software for their hardware grew. The greater the range of functions, the greater the incentive for institutes and organisations to buy it.

126 One book that interested people should read if they want to learn quickly and thoroughly is »The Software Rebels« by Glyn Moody. A second is Sam Williams' biography »Free as in Freedom. Richard Stallman's Crusade for Free Software«. The following presentation is inspired by – without individual references but with explicit thanks – these two works.

At the end of the 1960s, the tide turned: IBM was forced by a lawsuit from the US Department of Justice to decouple hardware and software. Software was given its own value. This decoupling laid the foundation for a separate market. Operating systems and programmes became the relevant differentiators of the new business line. Hardware »only« delivered pure computing power. Software became a tradable commodity.

During this time, the first version of UNIX was developed at AT&T's Bell Laboratories, primarily for internal use. However, AT&T gave its Unix to the universities at cost price, including the source code. The fact that this code base was easily accessible first increased user interest. And this encouraged further developments, as did – ironically – the lack of support: AT&T itself offered no support for the product. So it was a matter of helping oneself, primarily and best of all in exchange with like-minded people.

Local users quickly joined forces, exchanged solutions, and improvements via preliminary versions of the Internet and helped with problems. Through this free, unorganised cooperation in the academic community, UNIX quickly reached a high level of maturity.

A **first step towards institutionalisation** was then taken by the University of California at Berkeley. It gathered the emerging improvements, collected the new Unix tools and applications and published everything together continuously in its Berkeley Software Distribution (BSD). AT&T, on the other hand, turned to the task of distributing stable releases with commercial interest. However, when **1984** the fall of the telephone monopoly led to the break-up of AT&T, the successor company also took up the concept of software that had value on its own. Users of the tools from BSD now also had to purchase an AT&T source code licence, which became increasingly expensive. The commercialisation of Unix progressed, not without creating a counter-movement. Thus the restriction of free use via the introduction of licence fees also led to the fact that – after a corresponding call in a concerted action by BSD developers – remaining AT&T code was completely removed from the BSD Unix derivative. Over time, a truly free Unix thus emerged. The possility to access (besides Linux) OpenBSD, FreeBSD etc. as a free Unix, is due to this development.

The commercialisation of Unix progressed, not without creating a counter-movement.

9.1.2 A second and different path led to Linux

Also at Richard M. Stallman's university – MIT's Artificial Intelligence Laboratory with its original »hacker« and »sharing« culture in software – the development of commercialisation and compartmentalisation was practically noticeable. A popular anecdote is that Richard Stallman once wanted to adapt his printer to new interfaces, but the necessary printer software was not included in the printer's source code. Even when Stallman finally found a university colleague who had this code, the colleague was not allowed to pass it on to Richard Stallman because of the licence agreement. Software suddenly had a value of its own that had to be protected by exclusion. And it was precisely this exclusion – according to the simple view of Richard M. Stallman – that hindered him in his activities.

This kind of protection by turning away from the idea of free use soon extended beyond the commercial operating system Unix. Since patents on software became possible in the USA in 1981, there was also an increased desire among developers themselves to own programmes. From this side, too, the willingness to freely exchange software was thus limited.

Stallman felt both developments to be a personal limitation. And thus, starting in **1984**, he tried to revive the original dynamics of a culture of sharing, as he had come to know and appreciate in the early UNIX and BSD community at MIT. He began to develop and distribute the **GNU operating system** (a recursive acronym: »GNU«s not »UNIX«) under the terms of free software. Even more: he conceived the idea of free software and thus became the forerunner of the GNU project together with the Free Software Foundation (FSF), which was founded from it in 1985. GNU was to be a completely free and open operating system.

Free not as in the sense of »free beer«, but in the sense of »freedom«. It was to include a set of applications that would allow any user to use, view, modify, and redistribute the software in modified form without restriction.[127] This desire gave rise to the GNUGPL, the GNU General Public License, which for the first time cast the concept of »free software« into a licence.

Numerous applications quickly emerged within the GNU project. The guaranteed freedoms (re-)established (university) cooperation through exchange. Stallman's idea worked: the cooperation of individuals on a small scale could be worthwhile because the system as a whole would be permanently freely available.

However, the GNU project could only live up to its claim of creating a completely free, complete unix operating system with its own kernel. This central component of every operating system connects the modules and makes them executable. As long as a commercial Unix kernel was still needed to use all the free tools of the GNU project, the goal was not achieved. It was Linus Torvalds who finally programmed this kernel at

127 cf. ↗ https://www.gnu.org/philosophy/free-sw.html

the beginning of the 1990s, called it Linux and has had a decisive influence on its development to this day. He made his Linux kernel available as free software right from the start by distributing the code under the terms of the GPL. And there, too, Richard Stallman's idea bore fruit: many more computer scientists around the world were inspired to participate.

Thus, in combination with the already completed applications of the GNU project, what is known as LINUX today was eventually created.[128]

9.1.3 From »Free« to »Open«

The **impetus for the emergence of the open source movement** was **the disclosure of Netscape Navigator** at the end of the 1990s. At that time, Netscape was unable to assert itself with its browser against Microsoft and the dominance of Internet Explorer. Therefore, Netscape's code base was to be released and the browser was to be maintained by a free community. Today we know the result as »Firefox« and the associated Mozilla project.

But at the time of this release of an initially commercial product, the attribute »free software« proved difficult in communication with companies if one wanted to pursue new commercial business ideas on this basis. The philosophy associated with »free software« and the now common understanding of code as a company secret had a deterrent effect. In order to get into better contact with management and decision-makers in corporations, an alternative name was sought. It should sound more business-friendly and be less ideologically tainted. The suggestion to use the name »Open Source Software« for it is said to have come from Christine Peterson (Foresight Institute). Based on this name, the **Open Source Initiative (OSI) was finally founded in 1998** by Bruce Perens, Eric S. Raymond and Tim O' Reilly, which brought together ideas of free software and the Debian Free Software Guidelines to form the Open Source definition.[129]

In terms of usage, there is no difference between free software and Open Source Software: free software licences are almost all also listed as open source licences.[130] Conversely, the four constitutive freedoms of free software – meaning: to be allowed to use, understand, distribute and improve it[131] – are completely covered by the open source definition. The differences manifest themselves in the messages of the terms:

»Open Source Software« emphasises the practical benefits and the development method; »free software« emphasises the social benefits.[132] Put pointedly, non-free

128 The entire operating system is also referred to as GNU/Linux to emphasise that an operating system does not only consist of its kernel and that the tools from the GNU Project have an equally important part in the functioning whole. (Cf. GNU Project: Why Free Software is Better than Open Source Software; ↗ https://www.gnu.org/

129 Open Source Definition ↗ http://opensource.org/osd

130 cf. OSI: Licenses by Name ↗ http://opensource.org/licenses/alphabetical

131 cf. GNU Project: What is Free Software? ↗ https://www.gnu.org/philosophy/free-sw.html

132 cf. Stallman, Richard: Why Open Source misses the mark Free Software, ↗ https://www.gnu.org/philosophy/open-source-misses-the- point.html

software is »[...] for the open source movement [....] a suboptimal solution [;] for the free software movement [... but] a social problem and free software (its) solution.«[133] In any case, when one thinks of free software, one should »[...] think of free as in freedom of speech, not as in free beer.«[134]

Occasionally, attempts are made to reconcile the name conflict via hybrid terms such as »Free/Libre Open Source Software (FLOSS)« and »Free and Open Source Software(-FOSS)«.[135]

This fits with the statement that »the free software and open source movements [...] are something like two political camps within the free software community.«[136]

9.1.4 Generation GitHub

The first decades of open source were marked by the struggle for the new idea of free and open software. Many people and companies did not take the concept of the four freedoms seriously, saw it as too idealistic or even actively fought it.

> Linux's jokingly proclaimed goal of »world domination« has become reality.

In the second half of the 2010s, however, this picture changed fundamentally. The idea of open source caught on and became mainstream, not only among enthusiasts and private idealists, but also across the breadth of the economy.

Two acquisitions from 2018 made this abundantly clear. IBM acquired the open source manufacturer RedHat for 34 billion US dollars and Microsoft bought GitHub for 7.5 billion US dollars. This was particularly noteworthy because GitHub, as a platform with 100 million repositories and over 30 million users at the time,[137] was effectively the home of the global open source community, and it marked the result of a radical turnaround for Microsoft from being an opponent of open source to being the company with the most open source contributors globally by comparison.

133 GNU Project: Why Free Software is Better than Open Source Software, ↗ https://www.gnu.org/philosophy/free-software-for- freedom.en.html
134 GNU Project: What is Free Software?, ↗ https://www.gnu.org/philosophy/free-sw.html
135 cf. Stallman, Richard: FLOSS and FOSS, ↗ https://www.gnu.org/philosophy/floss-and-foss.html
136 Siehe ↗ https://github.blog/2018-11-08-100m-repos/
137 Linus Torvalds referred to Linux as »just a hobby, won't be big and professional like gnu« in his first announcement email in the comp.os.minix newsgroup (see ↗ https://en.wikipedia.org/wiki/History_of_Linux)

The dominance of Open Source Software in all areas where software is an essential component can also be seen in the development of Linux. Started as a hobby project[138] in 1991, today, in 2021, it runs on billions of devices, from phones[139] to cloud servers[140] or super computers[141] to Mars helicopters[142]. Linux's jokingly proclaimed goal of »world domination« has become a reality.[143]

In many ways, open source has become taken for granted. Nadja Eghbal, in her 2016 research report for the Ford Foundation ↗ »Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure«, describes how open source is a large part of the digital infrastructure, and in doing so, like roads and bridges, can be seen as part of our public infrastructure. It also describes the challenge that can arise in maintaining this shared infrastructure.

In the youngest generation of female software developers, who take open source for granted, who consume code from GitHub without much fuss, the struggles of the past are no longer present.

The fact that it takes considerable effort to maintain this code is sometimes forgotten. Solving this problem is one of the challenges of open source for the future.

One can discuss and speculate a lot about what has caused this enormous success of open source. Many aspects are described in this guide. Ultimately, it can probably be seen as a mixture of natural development over time, hard economic advantages and the idealism of a fantastic community.

138 See ↗ https://www.theverge.com/2021/5/18/22440813/android- devices-active-number-smartphones-google-2021
139 See ↗ https://www.zdnet.com/article/linux-now-dominates-azure/
140 Since 2017, all 500 super computers on the TOP500 list run Linux: ↗ https://www.top500.org/statistics/overtime/
141 See ↗ https://spectrum.ieee.org/nasa-designed-perseverance- helicopter-rover-fly-autonomously-mars 142
142 See ↗ https://www.linuxjournal.com/content/25-years-later-interview- linus-torvalds
143 ↗ »Executive Order on Improving the Nation's Cybersecurity«

9.2 Software Bill of Materials (SBOM)

The term »Software Bill of Materials« or »SBOM« for short has

become increasingly present in recent years. It crops up in the context of software supply chains and questions about security, licence compliance and similar issues. With the US President's Executive Order 14028 of 2021[144] calling for SBOM as an element of enhancing cybersecurity, the concept has gained increased attention that is evident across the industry.

For Open Source Software, the concept of SBOM is of particular importance, as open source components now make up a large part of all software produced. According to the ↗ »Synopsys Open Source Security and Risk Analysis report 2022«, 97 per cent of all software contains open source components and, on average across all software, 78 per cent of projects consist of Open Source Software. The large number of open source components poses a particular challenge here. According to the report ↗ »Sonatype 2021 State of the Software Supply Chain«, there are a total of 37 million available open source components in the Java, Javscript, Pyton and .NET ecosystems alone.

In this chapter we want to introduce the concept of SBOM, explain how it can be applied and classify in which areas and for which purpose it is already relevant today and will become so in the future.

9.2.1 What is an SBOM and what purpose does it serve?

The term »Bill of Materials« comes from the physical world, where lists of pieces or materials are maintained for physical goods. This creates transparency about components of products and records data such as name, origin or manufacturer.

For example, in the event of manufacturing problems, this Bill of Materials gives manufacturers the opportunity to identify which components are affected, which products are affected and by whom the causes can be remedied.

144 This can be done, for example, by using ↗ tools of the SigStore project.

For consumers, the Bill of Material provides transparency about ingredients, such as the list of ingredients in food. This makes it possible to make a conscious choice of food, for example in the case of allergies.

Transferred to the software world, the Software Bill of Materials describes the complete components of a given software product or software component. In particular, data such as unique identification of a component, versions, licences and information about the structure such as dependency information are recorded.

In particular, dependencies can be a challenge, as they are often only implicitly selected, even small products can contain a large number, and without explicit analysis, surprising components can often emerge.

Once all components are captured via an SBOM, this enables the question to be answered: »What exact software am I actually using?« This provides the basis for carrying out activities such as:

- **Treatment of vulnerabilities.** Which applications are affected? What is the current patch status?
- **Licence compliance.** Which licences are included in the software? Are there incompatibilities or licences with undesirable effects?
- **Risk assessment.** Are components included that pose risks, for example in terms of security or support? How often are components with risks used? What is the impact of the risks?
- **Strategic overview.** What is the overall picture of software components in use? How can software components be strategically designed? For example, where is contributing to open source components valuable from a strategic perspective?

In order for SBOMs to fulfil their purpose, they must be maintained and made available together with the associated software artefacts. The responsibility for this lies with the creator of the respective artefact. This means that the availability of SBOMs must be ensured independently for self-created software. In the case of supplied software, the responsibility lies with the suppliers and must be demanded by the acceptors. If this is ensured throughout the entire software supply chain, the SBOM concept can develop its full effect.

9.2.2 What data is contained in an SBOM?

An SBOM contains a compact set of data to identify software components as well as the most important meta-data about each component, such as origin or licence. In addition, an SBOM contains document-level meta-information, such as the time and type of creation and what the SBOM refers to.

There are two important standards to describe SBOMs: SPDX and CycloneDX. They are described in detail in a later section. The minimum profile of SPDX defines the following data to be collected (Source: ↗ SPDX Specification 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

An SBOM always refers to a specific software artefact. This can be, for example, a software binary that is created for deployment in production, or a specific release of an open source project.

It is important that the SBOM for the given artefact represents a complete list of all software components contained. This is the only way to reliably answer questions about security or licence compliance, for example.

Open source components are an important aspect of SBOMs. They will dominate most SBOMs in number. But proprietary and self-developed components must also be covered to ensure completeness and to apply consistent approaches and tools.

Depending on the question, a different scope of SBOMs may be necessary. From a licensing perspective, for example, the list of all delivered components is often sufficient. From a security point of view, it may also be necessary to record components used in a building environment.

It must also be delineated which data belong in an SBOM and which are kept separately. Since the SBOM reflects the state of a software artefact, the life cycle of the SBOM should be the same as that of the artefact. This means that static information, such as licensing information, should be recorded in the SBOM, but dynamic information, such as what vulnerabilities are contained in the components, is better maintained separately.

9.2.3 How is incompleteness dealt with, how is it documented?

In some cases, it may be difficult, impossible, or not economically feasible to conduct a complete survey. This should be documented accordingly so that the »known unknowns« are also recorded. In the SPDX format, for example, there is the possibility to mark data fields as »NOASSERTION« to make it transparent that no licence information has been recorded (which is not the same as a component without a licence).

9.2.4 How is an SBOM generated?

In many cases, software contains a very large number of components. Even simple Javascript applications often have over 10,000 dependencies, and even in more conservative ecosystems like Java, applications often consist of hundreds of components.

This figure can only be handled economically through automation. In individual cases, a manually created SBOM may also serve its purpose, but complete coverage is only possible through the use of appropriate tools.

There is very active development in the area of SBOM tools, both by manufacturers of proprietary offerings and in the open source community.

Depending on the deployment scenario, language ecosystem and concrete software products, different solutions are available. It is recommended to look for pragmatic solutions that address the needs of development teams to ensure that SBOMs are actually generated and that the necessary effort is kept within limits. Due to the dynamic nature of the market and the rapid evolution of the topic, it is also recommended to maintain flexibility in terms of tools and focus on standard formats and APIs. More on this in the section ↗ Tools for managing SBOMs.

Existing and especially older software can pose a challenge here in SBOM creation. With modern production models, such as DevOps, it is quite easy to build appropriate tools into existing automation. With other models, this can be more difficult.

9.2.5 How is an SBOM transmitted?

SBOMs are generated as part of the software production process. In order for them to be usable for software consumers, they must be made available to them in a suitable way. There is still little standardisation for this beyond the data formats. To guarantee the integrity of SBOMs, it is advisable to protect them with digital signatures and to verify signatures accordingly when using them[145].

145 This can be done, for example, by using ↗ tools of the SigStore project.

9.2.6 How is a metric defined for the reliability and trustworthiness of a 3rd party delivery of an SBOM?

There is always the risk that an SBOM is incomplete or faulty – be it due to inadequacies or problems in SBOM creation or also due to deliberate manipulation. Therefore, it is necessary to assess the trustworthiness of supplied SBOMs and to ensure the reliability of self-generated SBOMs.

9.2.7 What standards and formats are used for SBOMs?

Two main standards for SBOMs have become established. The ↗ SPDX standard, supported by the Linux Foundation, which has also been recognised as the ISO standard ISO/IEC 5962:2021 since 2021, and CycloneDX, which originates from the ↗ OWASP community. There is also ↗ SWID, which, however, has a rather subordinate significance in the area of SBOM.

In addition to the data model, the standards define various exchange formats, such as JSON or RDF, so that efficient machine processing and exchange of SBOMs is possible. Tools also exist for converting between different formats.

In addition to the standards for the SBOM as a document, standards for individual elements are also relevant. One of these is ↗ SPDX Id, which creates unique identifiers for open source licences. The other is ↗ package URL, which can be used to uniquely identify software packages. Both standards are used in SPDX and CycloneDX.

9.2.8 How are SBOMs managed in the company?

SBOMs must be managed in the context of the associated software artefacts.
They unfold their benefits from two perspectives:

For an individual software artefact, they offer a way to make its composition and origin transparent. This can be used to answer specific questions about a particular piece of software. This can be useful for a development or operations team dealing with that software.

The aggregation of SBOMs for all software in use offers additional possibilities to answer central questions – for example, how strongly a company is affected by a certain software vulnerability or which dependencies exist with which software. In this way, audit processes, for example on the topic of licence compliance, can also be implemented and automated.

How SBOMs are managed in the company so that they can unfold their benefits depends strongly on the technical environment, processes and organisation of a company. The following issues play a role:

- Integration into CI/CD infrastructure
- Connection to asset management
- Specifications and recommendations for action as to when SBOMs must be generated or procured and in what form.
- Integration in ITIL
- Lifecycle management
- Standards and interfaces to store SBOMs and access the information they contain.

9.2.9 Analysing and visualising SBOMs

The machine-readable format of SBOMs makes it possible to effectively analyse or visualise the information they contain. This can be done through generic data management tools that work on the JSON format, for example, through specific scripts or through specialised tools for SBOM analysis.

9.2.10 Tools for managing SBOMs

There are a number of tools that can be used in the management of SBOMs. The market is very much in flux and a lot of development is taking place in both proprietary and open source tools.

No clear recommendation can be made as to which tools a company should use. This must be analysed and decided in the respective environment. The classification of tools according to different fields of application can be helpful (source: ↗ »SBOM Tool Classification Taxonomy«, NTIA SBOM Formats & Tooling Working Group): SBOM Tool Classification.

A tool overview from the perspective of the two SBOM standard formats can be found at ↗ SPDX-Tools and ↗ CycloneDX-Tools.

9.2.11 What should be considered when procuring software from suppliers in terms of SBOM?

Supplied software often represents a »black box«, where it is not recognisable how it was developed, which components are used and where these components come from. Open source licences usually require that at least licence texts are delivered with the software.

However, this usually does not allow conclusions to be drawn about the exact composition and versions of components. An SBOM can fill this gap.

It is not yet a matter of course that software is supplied with an SBOM as an »ingredients list«. Some software manufacturers are moving in this direction and provide SBOMs or comparable information on their own initiative. For this to happen across the board, however, software buyers must demand it on a broad scale.

In some areas, this is done through general requirements, such as the US Executive Order. It will make it mandatory to include SBOMs for software delivered to public bodies.

As a rule, however, it will be up to the individual design by companies how the availability of SBOMs can be enforced. This can be required as standard in contracts with suppliers, for example.

In the case of Open Source Software, the availability of the source code often makes it possible to create SBOMs oneself, so that information from manufacturers can be checked or missing information can be added.

A standard approach is defined in the ↗ OpenChain specification, which describes the handling of Open Source Software in the supply chain. It has also been available as standard ISO/IEC 5230 since 2021.

9.2.12 What should be considered when delivering software to third parties in terms of SBOM?

Making SBOMs mandatory in the delivery of software is a requirement for software producers. They must integrate the generation and delivery of SBOMs into their software creation and delivery processes. To ensure complete and accurate informationc, it is essential to automate these processes.

9.2.13 Classification of initiatives on SBOMs: What is happening in America, what is happening in Europe and especially in Germany?

The presidential decree 14208 mentioned at the beginning has triggered a strong preoccupation with the topic of SBOM in the USA. The perspective that software used in public administration must necessarily come with an SBOM puts pressure on software vendors. This is both a challenge and an opportunity to develop processes, procedures and tools for dealing with SBOMs and to bring them to a high level of maturity. This is creating a strong dynamic in the American market.

The topic is also very present in the open source community. Projects like Kubernetes have now made the creation of SBOMs part of their release process (See ↗ »We Built the Kubernetes SBOM and Now You Can Write Your Own!«, Adolfo García Veytia), there are a growing number of open source tools for dealing with SBOMs and the topic is also addressed in initiatives such as the ↗ Open Source Security Foundation (OpenSSF), founded in 2020.

One initiative that has very strong industry support is ↗ »The Open Source Software Security Mobilization Plan«, backed by OpenSSF and several dozen technology companies. Under this plan, $150 million will be made available to increase the security of the software supply chain. SBOM is one of the 10 »streams« of the plan.

At European level, there is the Network and Information Security (↗ NIS) Directive, which is the first approach to set cybersecurity standards at European level. An updated version (NIS2) is currently being voted on. It adds a promotion of open-source cybersecurity tools, in particular to give small and medium-sized enterprises access to adequate tools. However, the directive does not address the issue of SBOM.

One EU initiative is the ↗ Cyber Resilience Act, which is available in draft form. It directly addresses the security of digital products by obliging manufacturers to take security measures and providing consumers with transparency about security-relevant aspects of products. The draft provides for SBOM as a means to achieve this. An interesting aspect of the draft is that it exempts Open Source Software that is not developed in a commercial context from many obligations that are required in a commercial context.

9.2.14 Outlook: How will the SBOM issue develop?

The topic of SBOM has gained considerable momentum in recent years. It is widely discussed and there are many proposed solutions for various aspects. Much of this is still at an early stage of development and approaches still need to be tested and mature. However, there are already some examples where SBOMs are being used in a resilient way.

An SBOM in itself does not solve any problem, but it creates the basis for creating added value with standardised and automated methods. This can be the detection and elimination of vulnerabilities, the analysis and quantification of risks arising from the software supply chain, or the assessment and handling of compliance aspects, such as open source licences.

Since this requires the interaction of all the open source projects, manufacturers and consumers involved in software supply chains, it seems inevitable that standards will prevail that ensure interoperability of different tools and SBOMs from different sources. The community and industry are well advised to work in this direction.

The large number of open source components results in strong pressure for automation. The SBOM concept will only be successful if it is not a major burden in the development, delivery and use of software. This can be achieved through automation and good integration via standard formats and interfaces.

9.2.15 References

- ↗ SBOM page of the NTIA – Overview page of the US National Telecommunications and Information Administration on the subject of SBOM with quite in-depth basic material.
- ↗ SBOM page of CISA – Overview page of the US Cybersecurity & Infrastructure Agency with information on the implementation of the SBOM concept.
- ↗ The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness – Report of the Linux Foundation on the State of the Topic SBOM in the Industry
- ↗ Finding vulnerabilities with a SBOM – example of how SBOMs can be used to find vulnerabilities
- ↗ Awesome SBOM – Curated list of materials on SBOM.
- ↗ Software Component Transparency: Healthcare Proof of Concept Report – One of the early documents from which the idea of SBOM originated.

Appendix

List of abbreviations

AGB
General terms and conditions

ASP
Application Service Providing

BGB
Civil Code

BHO
Federal Budget Code

****Copyright Act**
Copyright and Related Rights Act

CPU
Central Processing Unit

FAQ
Frequently Asked Questions

FOSS
Free and Open Source Software

Gem HVO NRW
Ordinance on the Budgetary System of the Municipalities in the State of North Rhine-Westphalia

GWB
Act against Restraints of Competition

HGrG
Law on the Principles of Federal and State Budgetary Law

LHO NRW
State Budget Code of the State of North Rhine-Westphalia

LTS
Long Term Support

OSD
Open Source Definition of OSI

OSI
Open Source Initiative

OSS
Open Source Software

PatG Patent Act SäHO
Financial Regulation of the Free State of Saxony

SBOM
Software Bill of Materials

SPDX
Software Package Data eXchange

UWG
Unfair Competition Act

VOL/A
Vergabe- und Vertragsordnung für Leistungen – Teil A für Vergabe öffentlicher Auftraggeber bei Liefer- und Dienstleistungsaufrägen (Public Procurement and Contract Regulations for Services – Part A for Procurement by Contracting Authorities for Supplies and Services)

Bibliography and source list

[1] Benkard, Georg et al: Patent Act, Utility Model Act, Patent Costs Act; Commentary; 10. Aufl.; Verlag C. H. Beck, Munich 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 (eds.): juris PraxisKommentar Vergaberecht, 4th edition, Verlag Juris Saarbrücken 2013

[4] Institute for Free and Open Source Software Legal Issues (various authors): Die GPL kommentiert und erklärt, 1st edition March 2005.

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

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

[7] Laurent, Phillippe: »Open-Source-/Content Licenses before European Courts«, EOLE 2012; (on the internet at: ↗ https://www.slideshare.net/OpenWorldForum/12-foss-licences-before- courts-in-europephilippe-laurent-eole2012)

[8] MPEP: Manual of Patent Examining Procedure, Manual for Examiners of the US Patent Office, (on the Internet at: ↗ https://www.uspto.gov).

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

[10] Reincke, Karsten/Sharpe, Greg: Open-Source-License Compendium – How to Achieve Open-Source- License Compliance, Darmstadt, Bonn 2015;
(on the internet at: ↗ https://github.com/telekom/oslic)

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

[12] Van den Brande, Ywein/Coughlan, Shane/Jaeger, Till (eds.): The International Free and Open-Source Software Law Book, 2nd edition 2014; (on the internet at: ↗ https://github.com/IFOSSLawBook/ifosslawbook).

[13] Working Group Public Affairs of the OSB Alliance (author: Till Jaeger): Handouts on the use of EVB-IT when using Open Source Software, 2018 (on the internet at: ↗ https://osb-alliance.de/wp- content/uploads/2018/10/201805_OSBA_Handreichung_EVB-IT.pdf).

[14] Wuermeling, Ulrich/Deike, Thies: »Open Source Software: A Legal Risk Analysis«; in: Computer und Recht 2/2003, p. 87 ff.

Literature Additions

[1] Cluster Mechatronik & Automation e.V., Open-Source-Software, Leitfaden zum Einsatz in Unternehmen, 2nd extended edition, 2014 (on the Internet at: ↗ http://www.cluster- ma.de/publikationen/leitfaden-oss-2-erweiterte- ausgabe/index.html).

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

[3] international FOSS law review: ↗ http://www.ifosslr.org/ifosslr (Despite the lack of new publications since 2014, this site offers a wealth of legal information on Open Source Software).

Keywords glossary

R

S

Bitkom represents more than 2,200 member companies in the digital sector. In Germany, they generate around 200 billion euros in turnover with digital technologies and solutions and employ more than two million people. Our members include over 1,000 mid-size companies, 500 start-ups, and nearly all global players. They offer software, IT services, telecommunication or internet services, manufacture devices and components, are active in digital media, create content, offer platforms, or are otherwise part of the digital economy. Eighty-two percent of the companies involved in Bitkom have their headquarters in Germany, another 8 percent come from the rest of Europe, and 7 percent are from the USA. Three percent come from other regions of the world. Bitkom promotes and pushes the digital transformation of the German economy and supports broad societal participation in digital development. The goal is to make Germany a powerful and sovereign digital location.

Bitkom e.V.

Albrechtstraße 10
10117 Berlin
T 030 27576-0
bitkom@bitkom.org

bitkom.org

bitkom