

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

Martin Wagner
MASTER THESIS

JavaScript User Interface License Compliance Best Practices

Submitted on 24 June 2023

Supervisor: Prof. Dr. Dirk Riehle, M.B.A.
Professorship for Open Source Software
Faculty of Engineering, Department Computer Science
Friedrich-Alexander University Erlangen-Nürnberg

Declaration of Originality

I confirm that I have written this thesis unaided and without using sources other than those listed and that the thesis has not been submitted to any other examination authority and has never been accepted as part of an examination in the same or similar form. All content that has been taken from a third party, either verbatim or in essence, is marked as such.

Erlangen, 24.06.2023

License

This work is licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0), see <https://creativecommons.org/licenses/by/4.0/>

Erlangen, 24.06.2023

Abstract

In the context of the prevalent use of Free/Libre and Open Source Software (FLOSS), this thesis emphasizes the need to shift license compliance in JavaScript User Interfaces (UIs) from merely “being done, right?” to genuinely “being done right”. It underscores the collective responsibility of software developers, managers, and end users, showing the problem of non-compliance and insufficient attribution in JavaScript UIs and the potential legal implications if obligations are not met.

This thesis outlines a set of best practices within two workflows: the creation of a Software Bill of Materials (SBOM) and the composition of third-party legal notices. It demonstrates the feasibility of these best practices using the Open Source Software Review Toolkit (ORT) in large projects, while acknowledging its limitations and the need for specialized knowledge. It also considers FOSSology as an alternative tool.

Emphasizing the criticality of meeting both ethical and legal obligations, this thesis advocates for the application of the proposed best practices in license compliance for JavaScript UIs. It concludes with the unveiling of SCATool, a web application to simplify and automate the process of ensuring license compliance, highlighting the necessity of rigorous compliance strategies to keep up with today’s dynamically changing FLOSS landscape.

Acknowledgments

At this point, I would like to thank all those who supported and motivated me during the preparation of my master thesis. First of all, I would like to thank Professor Riehle, who supervised me during the entire course of my master thesis. I would like to express my sincere gratitude for the helpful suggestions and constructive criticism during the preparation of this thesis. Thanks also to all the diligent proofreaders. Special thanks also goes to my wife, who always supports me and was always there for me during this master thesis. Finally, I would like to thank my family, who made my studies possible through their support and also had my back during this thesis.

Legal Advice Disclaimer

This work is for informative and academic purposes only. The material provided herein is not legal advice.

Contents

1	Introduction	7
2	Fundamentals and Terminology	9
2.1	Definition of Free/Libre and Open Source Software	9
2.2	Software Licenses	10
2.2.1	FLOSS Licenses	10
2.2.2	Permissive Licenses	11
2.2.3	Copyleft Licenses	11
2.2.4	Proprietary Licenses	12
2.2.5	Public Domain	12
2.2.6	Dual Licensing	12
2.2.7	License Governance	13
2.3	FLOSS License Structure	13
2.4	License Compliance	16
2.5	Software Bill of Materials	16
2.6	Third-party Legal Notices	17
3	Problem Identification	19
3.1	Who is responsible for License Compliance in Software?	19
3.2	When do License Obligations come into Effect?	20
3.3	Distribution of Software	21
3.3.1	Third-party Boundary	21
3.3.2	Redistribution of Inbound Software	22
3.3.3	Differences by Use-Case and Form of Distribution	23
3.3.4	Making a JavaScript Website accessible is a Form of Distribution	25
3.4	Why is this Topic becoming more and more relevant?	25
3.4.1	Ever-increasing Popularity of JavaScript	26
3.4.2	How is License Compliance handled in JavaScript Web Applications?	26
3.5	Lessons from High-Profile Legal Disputes	27
3.6	About proper Attribution	29
4	Objective Definition	30
5	Solution Design	31
5.1	JavaScript User Interface License Compliance Best Practices	31
5.2	Composition of the third-party Legal Notice	33
5.3	Where to display Third-party Legal Notices to the User?	34
5.4	Exemplary Application of Best Practices	36
5.5	Implementation of the Best Practices at Scale	38
6	Demonstration	40
6.1	Setup of the OSS Review Toolkit	40
6.2	Generation of the Software Bill of Materials	41
6.3	Analysis of every individual component	42
6.3.1	Narrowing down the analyzed Path	43
6.4	Collection of Findings, resolution of Ambiguities, removal of false Positives	44
6.5	No Assertion Findings	45
6.5.1	NPM::node-forge:1.3.1	45
6.5.2	NPM::source-map:0.6.1 and NPM::source-map:0.5.7	46
6.6	Copyleft Findings	47
6.7	Inspection of License Choices and Patent Grants	47
6.8	Review of the detected Licenses	48
6.9	Generation of Third-party Legal Notices	49
7	Comparison to FOSSology	51
7.1	Setup of FOSSology	51
7.2	File Upload, Scan and Clearance with FOSSology	52
7.3	Comparison of ORT and FOSSology Results and their Limitations	53
8	Conclusion	55
	Appendix A Arbitrary Project Third-party Legal Notices	57
	References	60

Table of Figures

Figure 1: The software supply chain (Phipps & Zacchiroli, 2020).....	22
Figure 2: Websites that display legal notices out of the top 50 Websites in January accessed from IP addresses within Germany.....	27
Figure 3: Dependency tree of the arbitrary project.....	36
Figure 4: Lots of NOASSERTION findings in the QDAcity web UI.....	44
Figure 5: 47 rule violations remaining after usage of the ort-config repository.....	45
Figure 6: Different versions of the same software in one distribution can be found.....	46
Figure 7: Copyleft findings in the QDAcity web UI.....	47
Figure 8: Remaining rule violations after resolution of all ambiguities.....	48
Figure 9: FOSSology settings.....	52
Figure 10: Result overview shown after scanning the QDAcity war node_modules directory.....	53
Figure 11: The index of Microsoft.....	54

Index of Tables

Table 2.1: The Open Source Definition (Open Source Initiative, 2007).....	11
---	----

1 Introduction

In a world where copying is easy and information is free, the value of intellectual property comes from the license to use it, not from ownership of the underlying bits. – Lawrence Lessig

Most software is built using free, open-source components. But being free of charge does not necessarily mean free of obligations. Any open-source license consists of five parts: a copyright notice, a rights grant, obligations to fulfill, prohibitions, and a disclaimer.

These obligations can be as slim as *The Unlicense*, which allows users to freely use and modify software without any legal restrictions or requirements. This license effectively puts the software in the public domain.

Opposing to that, the obligations of open-source licenses can be as potent as with the *GNU General Public License Version 3 (GPLv3)*. The GPLv3 requires that any derivative works based on the original software also to be licensed under the GPLv3. Derivative work means the original software was modified in some way. It also requires the source code for the software to be made available to users. These obligations can pose challenges for organizations that seek to use GPL-licensed software, as they may not be able or want to comply with all the requirements. This is particularly true for organizations seeking to generate revenue from the software, i.e., software vendors.

As a result, some vendors may choose to use open-source software with less onerous licensing requirements in their products, even if it means paying for the software or sacrificing some benefits of using open-source components. Purchasing software does not in itself eliminate this challenge, but it can help in some cases. For instance, the vendor wants to use a software that would otherwise be licensed under the *GPLv3*, requiring the vendor to open source any modifications to that software. This particular software is also made available under a proprietary license, that does not have that obligation. Then a vendor may very well opt for buying the component.

However, even if these organizations select to tackle the challenge of proper attribution and license compliance, there often is no clearly defined way to overcome this challenge. One of the most common technologies used when displaying a web user interface is JavaScript, which is why this thesis focuses on JavaScript. JavaScript is a high-level, interpreted scripting language primarily used for creating web pages to provide interactive experiences to users.

Research questions

Thus, the main research questions for this thesis are:

1. *What are the best practices for license compliance in the context of JavaScript user interfaces?*
2. *How can these best practices be applied at scale?*

Answering these questions requires a thorough understanding of software licensing. Question 1 makes sure that different possibilities, as well as the current state, are being considered. Question 2 addresses the numerous tools that offer to help ensure compliance with all licenses

of a distributed software. These research questions are the foundation for the design of a solution to an existing problem.

Structure of this thesis

This thesis aims to define and demonstrate best license compliance practices for JavaScript user interfaces.

At the beginning, relevant basics and terms are explained. The explanation of Free/Libre and Open Source Software (FLOSS) and software licenses is followed by a deeper delve into various license categories, such as permissive, copyleft, proprietary, and the public domain. Afterwards dual licensing and license governance are explained, followed by the general structure of FLOSS licenses, the definition of license compliance, and the Software Bill of Materials (SBOM). It also provides an overview on third-party legal notices.

The subsequent chapter, *Problem Identification*, explains who is responsible for license compliance, its relevance, and when license obligations come into effect. Various aspects of software distribution are discussed, including the third-party boundary, inbound software redistribution, and distribution based on use-case and form of distribution. The chapter concludes with an analysis of the impact of JavaScript's popularity on license compliance, insights gained from notable legal disputes, and a note on proper attribution.

The *Objective Definition* chapter outlines the goal of establishing JavaScript user interface license compliance guidelines. The *Solution Design* section outlines how to create a SBOM, how to create a third-party legal notice file, and where to display these notices using *yjs* as a simple case study.

Following is a hands-on demonstration that illustrates how to apply these best practices on a large scale, demonstrating the setup procedure, how to create the SBOM using the Open Source Software Review Toolkit (ORT), and how to analyze each component included. The software analyzed in this context is the QDAcity Web User Interface, which will serve as an example. QDAcity is a cloud-based web application for multiple researchers to collaboratively conduct Qualitative Data Analysis (QDA). This thesis also discusses the handling of ambiguous or false positive findings, and the creation of the legal notice file, utilizing the analyzed project as an illustration.

Subsequently, the proposed method of using ORT is compared to an alternative tool, FOSSology, including its setup and scanning process, verifying the results and showing the viability of implementing the best practices at scale. Finally, a summary of the results of this work is given.

2 Fundamentals and Terminology

To navigate the domain of software licensing, a number of basic elements and terms are required. In order to facilitate the further elaboration of this thesis, these basics are explained in this chapter.

2.1 Definition of Free/Libre and Open Source Software

To fully comprehend the concept of **Free/Libre and Open Source Software (FLOSS)**, it is imperative to engage in a scholarly exploration into the historical trajectory of open-source software.

Initially, all free and open-source software was referred to as free software (Ballhausen, 2019).

Which means that the term *free software* needs to be defined first.

Free software, as defined by the Free Software Foundation (2019), grants users the *four essential freedoms*, which include running the program for any purpose, studying and modifying the source code, redistributing copies, and distributing modified versions. According to the Free Software Foundation, free software and open-source software share similarities and often overlap, but their definitions and philosophies differ.

This is in accordance with Fogel (2005), stating *free software* emphasizes the ethical and social aspects of software freedom. Or to put it in the words of Richard Stallman, who founded the Free Software Foundation in 1985:

“It’s free as in freedom — think ‘free speech’, not ‘free beer’.” — Richard Stallman

In 1998, the phrase **open source** was coined with the intention to clarify that the software is not necessarily free of charge. Open-source software (OSS) rather offers users increased adaptability, especially due to the accessibility of its source code. Sharing many similarities, both *free software* and *open-source software* essentially encompass the same licensing conditions. When the aspects of both are meant to be covered, the term **FOSS**, or **Free and Open Source Software**, is used (Ballhausen, 2019).

Ballhausen (2019, p. 82) further states, that “FOSS licenses are distribution terms for software. Such terms are necessary because software is protected under copyright laws. [...] Unless rights of use are granted, [...] third parties are not in a position where they can lawfully use the software.” And in all cases, failing to comply with the obligations of any license will revoke any rights of use grants. This is one of the main legal motives for this master thesis. Noncompliance with license obligations is a copyright law breach. However, distributing a product that is not license-compliant is a breach of contract, as shown in chapter 3.5.

The fact that rights to use have to be explicitly granted applies to all licensed Software, be it FOSS licensed or otherwise, for example proprietary licensed. However, if software is licensed as FOSS, the distribution terms meet certain requirements (see chapter 2.2.1).

According to the Open Source Initiative (2007), some people prefer to use the term **FLOSS (Free / Libre and Open Source Software)** to emphasize on the point that FOSS is not free in a sense of being gratis. FLOSS and FOSS can be used interchangeably. For the remainder of this master thesis, the term FLOSS will be used.

2.2 Software Licenses

The world of software licensing is diverse, with various license types serving different purposes and offering unique sets of rights and obligations. It always boils down to a grant of the rights to use the software in certain ways. Ballhausen (2019) describes it very accurately:

Software licensing means that rights of use to computer programs are granted. Such rights may be granted in various ways, including simple/single rights of use with all others having the same privileges or exclusively so that the licensee is the only one who may lawfully exercise a particular right. A license may be territoriality restricted (for example, the European Union only) or granted worldwide; it may be restricted in time (such as what happens with hardcover books before they become available as paperbacks), or it may be perpetually granted. Finally, licenses may be granted for varying types of use, including reproduction in whole or in part; translation, adaptation, arrangement, and other modifications; and distribution of a computer program and making software publicly available. (p. 83)

This chapter aims to provide an in-depth analysis of the structure of software licenses, delving into the distinct types and their characteristics. The examination of FLOSS licenses, proprietary licenses, and the public domain will provide insight into their respective subcategories.

2.2.1 FLOSS Licenses

FLOSS licenses grant users the four essential freedoms to run, study, alter and redistribute the software, subject to specific terms and conditions (Rosen, 2005).

The Open Source Initiative (OSI) provides the *Open Source Definition*, which lists ten criteria (shown in Table 2.1) that a license must meet to be considered open source (Open Source Initiative, 2007).

These are the principles that need to be fulfilled in order for the OSI to evaluate and formally declare that a license is indeed an open-source license. The actual obligations and legally important observations though can only be made from the respective licenses themselves. The OSI maintains a list of OSI approved licenses¹. A categorization of FLOSS licenses can also be found at the *Institut für Rechtsfragen der Freien und Open Source Software (ifrOSS)*², the ifrOSS is a private institute aiming to follow the rapid development of OSS from a legal perspective. Another list of FLOSS licenses commonly found in software is available from the Linux Foundation's Software Package Data Exchange (SPDX) project³.

FLOSS licenses can be broadly categorized into two groups: permissive licenses and copyleft licenses. All licenses from both categories fulfill the criteria of the Open Source Definition and grant users the four essential freedoms of free software.

1 <https://opensource.org/licenses/>

2 <https://ifrOSS.github.io/ifrOSS/Lizenzcenter>

3 <https://spdx.org/licenses/>

Table 2.1: *The Open Source Definition (Open Source Initiative, 2007)*

1. Free Redistribution	The license shall not restrict any party from selling or giving away the software.
2. Source Code	The program must include source code and allow distribution in source code as well as compiled form.
3. Derived Works	The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. Integrity of The Author's Source Code	The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time.
5. No Discrimination Against Persons or Groups	The license must not discriminate against any person or group of persons.
6. No Discrimination Against Fields of Endeavor	The license must not restrict anyone from making use of the program in a specific field of endeavor.
7. Distribution of License	The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. License Must Not Be Specific to a Product	The rights attached to the program must not depend on the program's being part of a particular software distribution.
9. License Must Not Restrict Other Software	The license must not place restrictions on other software that is distributed along with the licensed software.
10. License Must Be Technology-Neutral	No provision of the license may be predicated on any individual technology or style of interface.

2.2.2 Permissive Licenses

Permissive licenses, also known as non-copyleft licenses, allow users to use, modify, and distribute the software with minimal restrictions. Code that is permissively licensed may go "closed" and further developments can be made under a proprietary license. Examples of permissive licenses include the MIT license, the Apache license, and the BSD license. These licenses generally require minimal obligations, such as retaining copyright notices and disclaimers within the source code. They provide a high degree of flexibility, allowing the incorporation of open source code into proprietary software without requiring the release of the proprietary software's source code (Laurent, 2004, Chapter 2).

2.2.3 Copyleft Licenses

Copyleft licenses, also referred to as "reciprocal" or "restrictive" licenses, require that any derivative work or modification to the software must also be distributed under the same license. This ensures that the freedoms granted by the original software are preserved in any subsequent works. The GNU General Public License (GPL) is a well-known example of a copyleft license. In addition to the obligations of permissive licenses, copyleft licenses often require that the source code for any modifications or derivative works be made available alongside any binary distribution of the software. A binary distribution is a version of the software that's

intended to be run on a computer, rather than human-readable source code (Laurent, 2004, Chapter 3).

2.2.4 Proprietary Licenses

Proprietary licenses are characterized by the fact that the software's source code is not openly accessible, and the right to use, modify, and distribute the software is restricted as Zittrain (2004) says:

For proprietary software not in the mass market, any number of arrangements might be agreed upon between the vendor and the consumer. The user might be permitted, for example, to see the source code and make changes, but not to distribute those changes to others. (Without such permission, consumer changes, if substantial enough, would comprise derivative works—the creation of which is a right reserved to the original author.) (p. 270)

The field of proprietary licenses is vast and could certainly be categorized into finer categories like e.g., commercial licenses, shareware licenses, and more. But since the use of proprietary software is usually associated with costs or at least with individual license-specific obligations, a finer categorization is foregone here.

2.2.5 Public Domain

Software in the public domain refers to software that is not subject to any copyright restrictions. When software is in the public domain, it is freely accessible to anyone and can be used, modified, and distributed without any restrictions or obligations (Laurent, 2004).

It is worth mentioning that placing software in the public domain is a deliberate act by the copyright holder, who must explicitly relinquish their rights to the software. Interestingly, you cannot simply get rid of the copyright in your work, since copyright in intellectual works are temporary ownership rights that expire over time. However, you can license software to the public (Rosen, 2005).

In some cases, software may be distributed under a permissive license that closely approximates public domain status, such as the Creative Commons Zero (CC0)⁴ or the Unlicense⁵. These licenses grant users the same freedoms as if the work were in the public domain, to the legal extent possible. Although these licenses serve a similar purpose, they should not be confused with the public domain itself, as they still involve a licensing agreement between the author and the user.

2.2.6 Dual Licensing

Software projects often have dual or multiple licenses. The owner of a copyright can license their own work as often as they want. This allows companies that produce proprietary software to open source parts of their software and FLOSS developers to sell their FLOSS licensed software under a proprietary license that grants certain warranties. In most cases, it allows licensees to choose from one of the provided licenses. In some cases, all licenses must be complied with at the same time (Rosen, 2005).

4 <https://creativecommons.org/share-your-work/public-domain/cc0/>

5 <https://unlicense.org/#the-unlicense>

2.2.7 License Governance

The license categories defined in chapters 2.2.1 through 2.2.6 can be used in a process called license governance to generally block material licensed under certain license categories from entering the software landscape. License governance encompasses the creation of a rule set that determines whether certain licenses can be used or not (Riehle & Lempetzeder, 2014).

I would advise against using abstract license categories to allow licenses without ever reading and understanding them, since for allowing a license the individual license content is crucial.

One template to help with setting up an effective open-source compliance program including license governance processes can be the specification provided by the OpenChain Project⁶. The OpenChain Project is overseen by the Linux Foundation and brings together various stakeholders, including open-source software developers, distributors, and users. It aims to reduce legal and operational risks associated with open-source software, promote trust between organizations, and streamline the process of open-source license compliance by providing a standard for open-source license compliance programs. By following the OpenChain Specification, organizations can ensure that they are using open-source software responsibly and in compliance with relevant licensing requirements (International Organization for Standardization, 2020).

However, license governance is beyond the scope of this master thesis, which focuses on developing best practices for license compliance in JavaScript user interfaces. From this point forward, I presume that the process of license governance has already been carried out and that the software project under consideration is aware of the types of licenses it utilizes.

2.3 FLOSS License Structure

The previous analysis of the different license categories shows that it is crucial for both developers and users to understand the structuring of software licenses to navigate the complex landscape of software rights and obligations.

The most commonly used category of licenses is the permissive license category, since permissive licenses allow for easy reuse and redistribution, mainly under only one obligation: Providing proper attribution.

The prior chapter has provided an overview of the various types of software licenses, including open-source licenses, proprietary licenses and the public domain. By acknowledging the nuances of each license type, developers and users can make informed decisions about the software they create, use, or distribute, thereby ensuring compliance with the relevant legal frameworks. However, a broad understanding of software licenses doesn't replace the need for detailed scrutiny of specific licenses.

Reading license texts can be onerous, but knowing the common structure of all FLOSS licenses makes reading and understanding the obligations that come with certain licenses much easier. A FLOSS license, as has been shown in the prior chapters, sets the legal framework for the distribution and use of open-source software. These licenses are crucial to ensure the free-

⁶ <https://www.openchainproject.org/>

dom of software usage, modification, and sharing, while also protecting the rights of the authors. A FLOSS license typically consists of the following five parts:

1. Copyright notice

The copyright notice is a statement that asserts the author's ownership of the intellectual property rights in the software. It often includes the author's name (or the name of the organization), the year of creation, and occasionally a brief mention of the rights reserved under the applicable copyright law. It is highly unlikely that an 'All rights reserved' snippet would be found in a MIT license, as the copyright holder will subsequently explicitly grant certain rights. This copyright notice is essential to inform users about the legal status of the software and to ensure that the author's rights are recognized.

Example: "Copyright (c) 2023 Martin Wagner"

2. Rights grant

This part of the license outlines the specific rights granted to the users of the software. These rights typically include the freedom to use, modify, distribute, and in some cases study the software. The rights grant ensures that users can enjoy the benefits of open-source software, encouraging collaboration, innovation, and the sharing of knowledge.

Example: "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."

3. Obligations to fulfill

This section specifies any obligations that users must fulfill when exercising the rights granted under the license. Common obligations include preserving the copyright notice, providing the source code when distributing the software, and attributing the original author when creating derivative works. These obligations help maintain the integrity of the original software and ensure that the contributions of the author and other developers are acknowledged. Failure to comply with these obligations may result in end users of your software taking legal action on the grounds of a breach of contract. Some examples on legal actions will be given in chapter 3.5.

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

4. Prohibitions

Prohibitions are restrictions placed on the use of the software. These limitations may include prohibitions on using the software for certain purposes (e.g., military applications) or restrictions on combining the software with other software that is not licensed under compatible terms. Prohibitions aim to prevent potential misuse of the software and to maintain compatibility with other open-source projects.

Example: “You may not use the software for any purpose that is unlawful or prohibited by this License Agreement.”

5. Disclaimer

The disclaimer is a statement that limits the liability of the author and any contributors to the software. It typically includes a warranty disclaimer, which specifies that the software is provided “as-is” and without any warranty of any kind, either expressed or implied. The purpose of the disclaimer is to protect the authors from any legal claims arising from the use of the software, thus encouraging developers to contribute to open-source projects without the fear of potential legal repercussions.

Example: “THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.”

If you are familiar with the MIT⁷ license template, you may recognize some of these examples. The MIT license contains a Copyright notice, Rights grant, Obligations to fulfill, and a Disclaimer. It does not include any Prohibitions. A prominent example of a license very similar to the MIT license that does indeed contain a prohibition is the JSON license⁸. The prohibition stated is “The Software shall be used for Good, not Evil.” However, this prohibition can hardly be interpreted without further clarification from the author of the software, putting the JSON license on the mental “difficult-to-comply-with” list.

Typically, not every FLOSS license contains all five parts, and some licenses may contain more sections than those I outlined. However, this breakdown makes it easier to read any license that you may not be familiar with yet, since it gives you an idea as to what to expect from a license.

⁷ <https://mit-license.org/>

⁸ <https://json.org/license.html>

2.4 License Compliance

Larger companies have realized that being in compliance with all involved licenses is anything but trivial. The use of approval processes is needed to ensure license compliance in a company landscape, as illustrated in chapter 2.2.7.

A legally valid software product needs to be in compliance with the licenses of all components included in the product. Failure to do so leads to a legally invalid product. Open-source licenses provide such rights only if and while certain obligations are fulfilled (Meeker, 2017).

Violating some of these obligations can lead to intellectual property loss for the software vendor. A violation of a license obligation creates a problem that cannot be resolved easily “post mortem”. If you released and published software that is not license-compliant, there is no easy way to “fix it”. You can fix it for the current version, but the old version of the software can still be subject to legal action (Ruffin & Ebert, 2004).

In the context of this master thesis, pursuing *license compliance* means that all obligations of the licenses involved are met and at the same time no prohibitions are violated. These obligations differ by use-case and by form of distribution, as discussed in further detail in chapter 3.3. This is due to the nature of the various FLOSS licenses.

2.5 Software Bill of Materials

The Executive Order on Improving the Nation’s Cybersecurity was the origin of the common practice of federal agencies to require a Software Bill of Materials (SBOM) when acquiring software from their suppliers (National Institute of Standards and Technology, 2021).

The SBOM is a “formal record containing the details and supply chain relationships of various components used in building software” (The White House, 2021).

Minimum Elements for a SBOM are a supplier (manufacturer) name, component name, the version of the component, other unique identifiers, a component’s dependency relationship, the author of the SBOM data and a timestamp. One of the recommended data fields is license information (The United States Department of Commerce, 2021).

In other words, it is a software parts list that includes each FLOSS component, and its identified licenses from which the supplied software is comprised.

Federal agencies require suppliers to adhere to provide such a SBOM. Furthermore, suppliers must ensure that the SBOM conforms to industry standard formats to enable automated ingestion and monitoring of versions. Current accepted standard formats are SPDX⁹, CycloneDX¹⁰, and SWID¹¹, as suggested by the National Telecommunications and Information Administration (National Institute of Standards and Technology, 2022, 2022).

Out of the three SPDX and SWID are also released as ISO standards (International Organization for Standardization, 2015, 2021).

9 <https://spdx.dev/>

10 <https://cyclonedx.org/>

11 <https://csrc.nist.gov/projects/Software-Identification-SWID>

Other standards define processes and compliance programs. These standards provide a more abstract view on the workflow that is needed to achieve a license-compliant product. To support the systematic review and approval of each component’s license terms, a bill of materials is required. This bill of materials helps to understand the obligations and restrictions that each license applies to the distribution of the entire software (International Organization for Standardization, 2020).

In summary, the SBOM is the initial critical artifact in the license compliance process due to its comprehensive detailing of software components and their corresponding licenses. It aids in maintaining transparency, ensuring compliance with standards, facilitating systematic reviews of license terms, and ultimately helping comprehend the obligations and restrictions applicable to the distribution of the entire software.

2.6 Third-party Legal Notices

A *legal notice* is a formal document used for initiating communication between two parties/persons. It is used to inform one party of their grievances with another party and their intention of taking legal action against them. The purpose of sending a legal notice is to give appropriate time to the receiving party/person to resolve the conflict.^{12 13 14}

This does, however, not apply to the *third-party legal notices*, according to Riehle & Harutyunyan (2019). Third-party legal notices are a display of copyright notices, declared licenses and license texts for the component. Riehle & Harutyunyan further explain that a third-party legal notice is a file seeking to fulfill two of the most common obligations of FLOSS licenses:

- The license provision obligation
 - Provide the license of each component that comes with the product.
- The copyright notice provision obligation
 - Provide all copyright notices from all files beneath each component involved.

Some licenses mention this third-party legal notice file directly. According to Section 4 of the Apache license, Version 2.0 the following is required. If the work includes a “NOTICE” text file as part of its distribution, then any derivative works that you distribute must include a readable copy of the attribution notices contained within such NOTICE file. The notices must be included in at least one of the following places: within a NOTICE text file distributed as part of the derivative works; within the source form or documentation if provided along with the derivative works; or within a display generated by the derivative works. The contents of the NOTICE file are for informational purposes only and do not modify the license (The Apache Software Foundation, 2004).

12 <https://www.cleverism.com/lexicon/legal-notice-definition/>

13 <https://legodesk.com/legopedia/what-is-a-legal-notice/>

14 <https://restthecase.com/knowledge-bank/tips/what-is-a-legal-notice-and-how-to-send-it>

Android applications handle this slightly differently, but also by no means in a unified way. Matt Compton shows (2015), that the display of the third-party notices is mostly done in one of the following ways:

- Opening a browser and displaying a website containing the license information.
- Displaying a simple text dump of all copyright notices and license texts.

Some other examples for third-party legal notices can be found at Microsoft¹⁵, what3words¹⁶ or mapbox¹⁷. The third-party legal notice is typically grouped by package. This means that it contains a list of package names. For each package, all licenses found in the context of this package are listed. Furthermore, all copyright notices are listed under each license. In chapter 5.2 this will be explained in more detail.

It is unclear to me, as to why the display of these notices is sometimes grouped by license, as it is being done with what3words, rather than grouped by package. Notices that are grouped by license just display a list of licenses, and all copyrights are added to the license under which they occur. An MIT license in such a list would have a very long list of copyrights attached to it, due to its very common use. This obfuscates the actual origin of the copyright notices by detaching the copyright holders from their own work, and thus does not provide proper attribution.

In summary, in the context of FLOSS, a *third-party legal notice* refers to the display of copyright notices, declared licenses, and license texts for each software component. As of the time of writing this master thesis, there is no established standard for displaying these notices. It is typically just being done, usually either by directing users to a different website or providing a text dump of notices and license texts.

15 <https://www.microsoft.com/en-us/legal/third-party-notices>

16 <https://what3words.com/third-party-legal-notices>

17 <https://github.com/mapbox/mapbox-maps-android/blob/main/LICENSE.md>

3 Problem Identification

Having established the foundation in chapter 2, next is the identification of the problem that this thesis seeks to solve.

What are the best practices for license compliance in the context of JavaScript user interfaces?

The problem being that there are currently no well-defined best practices for license compliance for JavaScript user interfaces. There are approaches and standards on a more abstract level regarding license compliance in general, but they do not go into much detail about how to apply license compliance in a JavaScript context step by step. The best practices I have developed could also be applicable to other programming languages, but within the scope of this master thesis I cannot cover all of them and will therefore focus on the most commonly used programming language in the context of web user interfaces.

To answer that question and solve this problem, a couple of relevant aspects needed to be analyzed. Starting with the question of who is responsible for license compliance in software, resuming with the question as to when license obligations apply. Unraveling the definition of distribution of software and showing how license compliance is being handled as of the time of writing this thesis. This chapter continues with information on some recent lawsuits that might expedite the development of more and more companies striving to comply with the licenses of their included FLOSS, and concludes on a note on proper attribution.

3.1 Who is responsible for License Compliance in Software?

Laurent (2004) argues that everyone who redistributes software is responsible for license compliance. This includes vendors and anyone else who distributes software to third parties, as well as end-users on the receiving end.

Rosen (2005) does provide some insight into how companies should approach open-source licensing. He emphasizes the importance of understanding the various open-source licenses and their implications for the company's software usage and distribution. Rosen further explains the potential legal risks if a company is being sued for a license obligation breach. There are monetary risks, but also the risk of the court speaking an injunction. If the component in question is a critical part to the software of the sued company, an injunction could very well mark the end of their product. But he also states that they were unable to test their theories because the licensee in question had stopped using the code whose license they were breaching. In other words:

The ungoverned use of FLOSS components can result in legal problems resulting from inadequate license compliance, operational issues resulting from lengthy release reviews including scanning and documenting the used open source components, financial and intellectual property issues resulting in litigation, cease and desist claims or product recalls (Harutyunyan & Riehle, 2021).

Harutyunyan & Riehle (2021) further identify license compliance as one of the critical issues around the use of open-source components in products and suggest several best practices to

address this issue. These best practices include developing a FLOSS governance policy for the transition period, establishing a set of standards and tasks for employees to follow, and running random audits to identify potential license compliance issues, as explained in chapter 2.2.7.

In conclusion, license compliance in software is a shared responsibility among all parties involved in the distribution and usage of software, including vendors / distributors, and end-users. Companies should be aware of the implications of various open-source licenses and their impact on software usage and distribution, as ungoverned use of open-source components can lead to legal, operational, and financial issues. To mitigate these risks, it is essential to implement best practices, such as developing a FLOSS governance policy, establishing standards and tasks for employees, and conducting random audits to ensure license compliance. At the same time, end users should inquire and verify that the software they are placing their trust in is meeting its ethical and legal duties to comply with the licenses of the FLOSS components being used.

Regardless of the specifics of the role, it is certain that everyone involved in the distribution of software – be it a developer, manager, or end-user – shares the responsibility of being in license compliance and demanding it.

3.2 When do License Obligations come into Effect?

To understand when licensing obligations are triggered, the following statements were considered.

Laurent (2004) explains that the obligations of FLOSS licenses are commonly triggered by the act of distribution, which means making the software available to others in some way. For example, distributing a software program on a CD, making it available for download from a website, or providing it to others on a network are all forms of distribution that may trigger the license obligations.

In the case of collaborative work, Fogel (2005) notes that the license obligations become effective as soon as the contributor makes a contribution to the project. This is because each contribution is in itself a form of distribution. This means that contributors must ensure that their contributions are license-compliant, and the project maintainers must ensure that the project as a whole is compliant with all involved licenses. Failure to do so could result in legal risks for both the contributor and the project as a whole. In this context, often contributors are also asked to sign a Contributor License Agreement (CLA), which effectively “copies” the contribution in a legal sense. The project is free to use the contribution as they please, but the contributor also is still free to use their own contributions for any other purpose, as they remain the owner of the copyright. Developer Certificates of Origin (DCO) on the other hand also require the contributor to use the same license for their contribution as the outgoing project license.

Rosen (2005) notes that license obligations do apply when the software is distributed or made available to the public. This can happen in various ways depending on the specific license.

For example, the GPLv3¹⁸ requires that the source code is made available to anyone who receives the software, while the Apache 2.0¹⁹ license requires that any modifications to the software are clearly marked as such if you reproduce and distribute the software. Rosen also notes that the obligations of open-source licenses can extend to derivative works, depending on the specific license terms. This means that if someone creates a new software program that is based on or includes the open-source software, the license obligations of the original software may still apply to the new program. Most copyleft licenses do just that, as shown in chapter 2.2.3.

In summary, the obligations of most FLOSS licenses come into effect when the software is (re-)distributed or made available to others, either in its original form or as a modification. This can happen through various means. Using the software internally or modifying it for personal use, for example, often does not trigger the license obligations. The specifics of license obligations always depend on the terms of the license itself, and failure to comply with these obligations can result in legal risks for both individuals and organizations.

3.3 Distribution of Software

To understand what exactly the *distribution of software* means, an overview of the third-party boundary, the different use-cases and ways of distribution and what all of that means for JavaScript user interfaces will be presented.

In chapter 1, the five typical parts of any open-source license have been shown. As shown in chapter 3.2, obligations often only come into play, if the software at hand is distributed, or re-distributed. Rosen describes distribution in a very intuitive way:

We speak of distribution if software is being sold or copies are given away to others. It also includes arrangements where software of others is incorporated in a consumer or industrial product and this product is being sold. It may also include making the software available across a network for execution by others (Rosen, 2005).

Making the software available across a network for execution by others is precisely what is happening when a user opens up a JavaScript website. The JavaScript code is sent across the internet from the hosting Server to the client's computer, where it is executed. This means that obligations of licenses that a vendor of software has to comply with also apply for JavaScript websites. In this case, the vendor would be the person or company hosting the website.

As the vendor of a website is distributing JavaScript code to the user, they need to comply with the obligations of the licenses of the components they are redistributing. Why exactly that is the case becomes even more clear when understanding the third-party boundary.

3.3.1 Third-party Boundary

In general, the line that needs to be crossed for obligations to apply, can be referred to as the third-party boundary. Phipps & Zacchiroli define the third-party boundary in the context of an open source supply chain:

18 <https://www.gnu.org/licenses/gpl-3.0.en.html>

19 <https://www.apache.org/licenses/LICENSE-2.0.html>

Given that almost every enterprise software system contains open source software, where has it come from? How is it manipulated and assembled to produce internal systems? Who receives the resulting constructions? These questions define a software supply chain, which may involve a surprisingly long and broad sequence of entities on the inbound side and could include third parties on the outbound side, even if your business does not apparently trade in software. [...] The open source supply chain comprises inbound software — the open source and proprietary software entering your enterprise — together with its dependencies; in-house development, the adaptations you make to inbound software and the software you develop; and then outbound software, the software you pass to others, either under open source licenses or proprietary terms, as software, as software-implemented services, or embedded in hardware (Phipps & Zacchiroli, 2020).

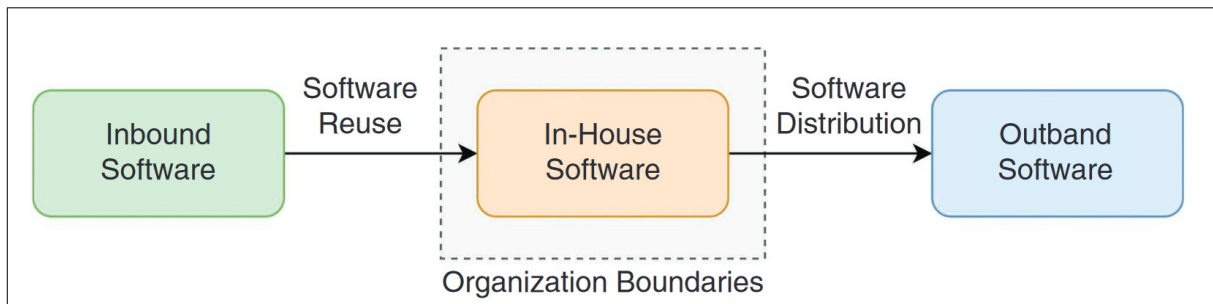


Figure 1: The software supply chain (Phipps & Zacchiroli, 2020).

Managing an open source supply chain will require a comprehensive open-source policy. Figure 1 shows this policy in a simple way. There is a clear distinction between *Software Reuse* and *Software Distribution*. For example, using a compiler for in-house applications is no distribution. The same applies to a demo of your software that you show to someone. The line that makes the difference is the third-party boundary. It marks the point when inbound software is passed on to a third party. Whenever this boundary (shown in Figure 1 by the dashed line labeled *Organization Boundaries*) is crossed by any outbound relation, distribution and redistribution of any inbound software takes place.

3.3.2 Redistribution of Inbound Software

The problems caused by open source arise primarily when unintentional open-source code ends up in the product source code. Or when it happens without the person importing the code being aware of the implications of doing so. There are several ways open-source code can end up in a product. Three main methods are manual copying, embedding FLOSS, and embedding FLOSS through third-parties.

The first instance occurs when a developer manually copies segments of open-source code into the source code of the product for which they are responsible. This could potentially result in unwanted code making its way into the product. The second scenario is one where the developer purposefully embeds an open source component into the product. The third and final scenario involves an open-source component being introduced into the product indirectly, via third-party software. In this case, the developer might not even be aware that the open-source component has been embedded, as it forms part of another software solution that has been integrated into the product. This can lead to unforeseen complexities and potential issues further down the line (Riehle & Lempetzeder, 2014).

One noteworthy detail to these three different ways inbound software can find a way into a product is that manual copies inside the code are hard to identify (Haddad, 2016).

Embedding an open-source component is usually done via a package manager like npm or yarn in the JavaScript context, and is therefore easier to detect than code that was copied manually. But the entire matter becomes really problematic when considering embedding via third-parties. Redistribution does not only occur for software components you explicitly embedded but also for their dependencies and the dependencies of their dependencies and so on. For JavaScript, this means all packages that have been installed need to be considered. Since the more problematic part is software that is already licensed and over which a product owner has no power, for the rest of this paper the terms distribution and redistribution will be used interchangeably. When thinking about distribution, redistribution must be considered as well.

3.3.3 Differences by Use-Case and Form of Distribution

The two different use-cases that can be directly derived from the analysis of the third-party boundary are firm-internal or personal use and distribution to third parties. Often, license obligations only take effect when redistribution takes place. This motivates the distinction between the two main use-cases of software, being firm-internal, in-house or personal use and distribution to third parties, e.g., sale to customers.

In addition to that, there are different ways of distribution and depending on the licenses involved, the way of distribution will influence the obligations that need to be fulfilled. This is due to special amendments and exceptions to some licenses. The probably most prominent example is the GPL linking exception, as used in the OpenJDK Assembly Exception²⁰. The GPL linking exception allows certain pieces of software to be *linked* to GPL-licensed software, meaning they can be combined, typically by incorporating library calls to the GPL-licensed code from the other software. This linking can occur without making the entire combined software subject to the GPL, which usually requires that the derivative work be distributed under the same license, thus preserving the other software's original licensing conditions. Different license categories bring different obligations that typically trigger on these different forms of distribution, as has been found in chapter 2.2. Distinguishing between the following forms of distribution can help with both reading and understanding license texts, as well as in license governance.

It is worth mentioning that some licenses are agnostic to the form of distribution, such as the MIT license. It requires the preservation of the original license and copyright notices, independent of the form of distribution. The obligations of others, e.g., the GNU Lesser General Public License (LGPL) Version 3²¹, which includes a linking exception, are dependent on the form of distribution.

For a better understanding of what the differences between different forms of distribution can look like, the following example is considered. If a software which includes LGPL licensed code of a third party is distributed in the form of an unchanged binary, the following are some of the obligations. Prominent notice needs to be given, the GNU GPL and the LGPL need to

20 <https://openjdk.org/legal/assembly-exception.html>

21 <https://www.gnu.org/licenses/lgpl-3.0.html>

be distributed along with it and either the minimal corresponding source must be given or a shared library has to be used and linked to.

If the software was a derivative work and was distributed as a binary, you may distribute a modified version of the library under this license or the GNU GPL. When you create a derivative work and distribute it as a binary, the LGPL requires you to also distribute the source code for your modifications.

If you distribute the source code of LGPL-licensed software, it must be under the LGPL or the GPL. You have to include a copy of the LGPL with the software, and any modified files must carry notices stating that you changed the files.

As this example illustrates, distinguishing between the different forms of distribution can be a practical way to avoid forgetting specific aspects of certain licenses in particular scenarios. In summary:

Use-Cases

- Firm-internal or personal use
- Distribution to third parties

Forms of Distribution

- Unchanged binary distribution
- Derivative binary distribution
- Source code distribution

Another, more abstract way to define project rules is to use common FLOSS license use cases.

Common FLOSS License Use Cases

As suggested by the OpenChain Project, any project should be capable of dealing with the most common open-source license use cases. The OpenChain Project also advocates identifying and defining how to act on these common scenarios:

- Your product is integrated with other FLOSS such that it triggers additional licensing obligations
- Your product contains modified FLOSS
- It contains FLOSS or other software under an incompatible license interacting with other components within the Supplied Software; and/or
- Your product contains FLOSS with attribution requirements.

This list is neither exhaustive, nor might all of the use cases apply (International Organization for Standardization, 2020).

Combining the different forms of distribution, the different use-cases and the most common open-source license use cases allows companies to define rule-sets that need to be followed. A process which is known as license governance, a process explained in chapter 2.2.7.

However, some licenses also have obligations that may trigger on other conditions than distribution. One example is the GNU Affero General Public License Version 3.0 (AGPLv3)²².

AGPLv3 requires that if the modified program offers network interaction to users, the corresponding source code of the modified version must be made available to those users. Another example is End User License Agreements (EULAs), for instance the Microsoft Software License Terms²³. These terms explicitly restrict the use of the software. They forbid reverse engineering, decompiling, or disassembling the software, or attempting to do so, except and only to the extent that the foregoing restriction is (a) permitted by applicable law; (b) permitted by licensing terms governing the use of open-source components that may be included with the software; or (c) required to debug changes to any libraries licensed under the GNU Lesser General Public License that are included with and linked to by the software.

So unlike most open-source licenses, the Microsoft Software License Terms obligations are not dependent on distribution of the work. Instead, they apply whenever someone uses the software. At this point, few may still argue, they are only providing a website on the internet, how should that be considered distribution.

3.3.4 Making a JavaScript Website accessible is a Form of Distribution

All of the following scenarios are examples of software distribution. Making a web user interface (UI) available to third parties (as explained in chapter 3.3), making a mobile application (app) available through an app store or making a container image available through a repository. It also includes selling a physical embedded device with open-source code, making a firmware update available through the web, letting a customer download a software application and many more.

In this thesis, the focus is on JavaScript software, which is being distributed to clients. Whether this is considered code distribution or not is not apparent. It, however, clearly is a form of distribution. This contrasts with server applications, where only messages containing information are passed between client and server, but no actual distribution of the entire server application takes place.

In a typical scenario, the JavaScript UI that you distribute by making it available for clients through the World Wide Web needs to be license-compliant. The servers that your UI is communicating with, that are also developed by you and may or may not contain open-source components, do not necessarily need to comply with the same obligations as the UI. This is due to the server software not being distributed. However, if your server should contain modified AGPLv3-licensed code, you must also make the corresponding source code available to the site users. This is because the modified program provides network interaction to the users.

22 <https://www.gnu.org/licenses/agpl-3.0.en.html>

23 https://www.microsoft.com/en-us/UseTerms/OEM/Windows/11/UseTerms_OEM_Windows_11_English.htm

3.4 Why is this Topic becoming more and more relevant?

This chapter delves into the rising importance and relevance of legal notices in software, particularly in the context of JavaScript web applications. It shows the increasing popularity of JavaScript. Then the results of my analysis of popular websites are shown, which depict how license compliance is currently being managed within JavaScript web applications.

3.4.1 Ever-increasing Popularity of JavaScript

In recent years, there has been a significant increase in the use of JavaScript for building user interfaces in web applications:

2022 marks JavaScript's tenth year in a row as the most commonly used programming language (Stack Overflow, 2022).

The state of JavaScript is largely continuing the way trends would have suggested last year. We're shipping more of it, for sure [...] (Wagner, 2022).

Whilst at the same time, a significant increase in the use of open-source components can be seen (Deshpande & Riehle, 2008; GitHub, 2022).

As a result, there is a growing need for developers to understand how to comply with open-source licenses when using JavaScript libraries and frameworks.

JavaScript libraries and frameworks are usually distributed under open-source licenses, such as the MIT license or the Apache license. These licenses typically require developers to include a copy of the license and copyright notice in their software. Some are also licensed under open-source copyleft licenses, which require developers to make the source code available if they distribute the software. The source code provision obligation on modification is what classifies such licenses as copyleft licenses. A prominent copyleft license example that has already been mentioned in this thesis is the GNU General Public License (GPL).

Complying with these obligations can be challenging in the context of JavaScript user interfaces, where the code is regularly minified or obfuscated. Minification and obfuscation can remove or alter important information that is required to comply with open-source licenses.

For example, open-source licenses often require that the original copyright notice and license text be included with the distributed software. However, when code is minified or obfuscated, the original copyright notice and license text may be removed or altered. This can make it difficult to determine which open-source components are being used and whether they are being used in compliance with their license terms.

3.4.2 How is License Compliance handled in JavaScript Web Applications?

In order to gain a comprehensive understanding of the present circumstance, I conducted an analysis of the 50 most frequently accessed websites from IP addresses within Germany in January 2023 (*Top Websites Ranking in Deutschland in Januar 2023*, n.d.). Due to inappropriate content, only 46 out of the 50 websites have been analyzed.

It is often claimed that everyday practice of license compliance in JavaScript applications diverges from the obligations of the most commonly used open-source licenses, the MIT license (Synopsys Editorial Team, 2022; Zacchioli, 2022).

The outcomes depicted in Figure 2 speak for themselves. Only two websites did any form of attribution at all, and only one of them did provide a form of legal notices, that was incomplete. That is even though all the analyzed sites distribute some JavaScript code. That means if any of the analyzed websites use any MIT licensed package, they are violating the obligations of the MIT license. Namely, the obligation to provide a copy of the copyright notice and the permission notice in all copies or substantial portions of the software. A website using the code and functionality of an open-source component is a substantial portion of the software.

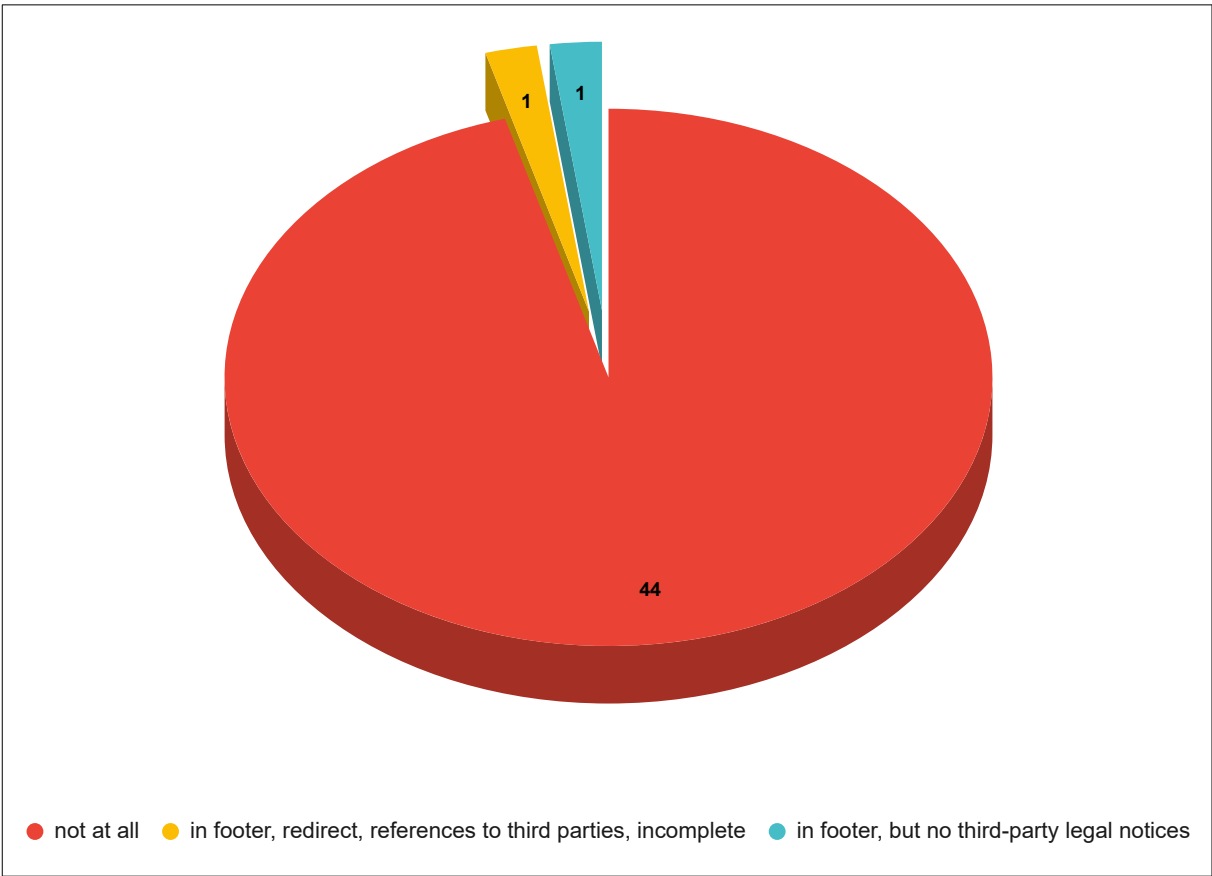


Figure 2: Websites that display legal notices out of the top 50 Websites in January accessed from IP addresses within Germany

By using the developer console of the browser and searching for the term “license” in 45 out of 46 websites, snippets of licenses have been detected. However, these are found within the source-code and not in a dedicated third-party legal notice. They are hidden from the users and not readily displayed. I explain why this is alarmingly problematic in chapters 5.2 and 5.3.

But is it really that problematic? Is there even a chance of me being sued at all? If these are questions that come to your mind now, then you might be interested in the following lessons from high-profile legal disputes.

3.5 Lessons from High-Profile Legal Disputes

The Software Freedom Conservancy's lawsuit against Vizio is a good example of the legal risks that can arise when open-source licenses are not properly complied with. In this case, Vizio was accused of violating the GPL by using open-source software in its TVs without complying with the license terms. While the lawsuit was ultimately remanded to California State Court, and is at the time of this writing still ongoing, it highlights the importance of complying with open-source license obligations (Software Freedom Conservancy, 2022).

One side effect of this is of particular interest to my research: It could very well be that the Software Freedom Conservancy is successful in demonstrating that third parties (i.e., receivers of the licensed code) have rights under the GPL. This could potentially encourage more receivers of licensed code to file complaints or take legal action if they believe that their rights under a copyleft license, or even any permissive license, have been violated.

This would mean that simply obtaining and using software gives you a right to sue. In this case, the initial complaint was for a breach of contract, the fact that a defective product was delivered to you. Therefore, any user would have the right to ask for compensation.

The court ultimately remanded this case to the state court. Before that, the court clarified the legal situation:

[The court] restricted what types of breaches may be construed as copyright infringement, but if anything, created a presumption that most breaches of licensing agreements will not only create a copyright claim, but even further, a breach of contract claim. If anything, the distinction between conditions and covenants only underscores that the disclosure obligation here is best characterized as a covenant actionable only under breach of contract (*Software Freedom Conservancy, Inc. V. Vizio, Inc. Et al*, 2022).

The Software Freedom Conservancy hopes to demonstrate that it's not just the copyright holders, but also the receivers of the licensed code, who are entitled to rights. The current development is heading in exactly this direction.

It is not yet legally proven, but if you want to be on the safe side, I recommend that you adhere to the practices described in the following chapters.

Another example of the importance of open-source compliance is the legal dispute between ChessBase and the Stockfish team. In this case, the Stockfish team accused ChessBase of violating the GPL by using the Stockfish chess engine in its software without complying with the license terms. The two parties eventually reached an agreement, but the dispute still highlights the potential legal risks of non-compliance with open-source licenses (VandeVondele, 2022).

In summary, compliance with open-source licenses is becoming more and more relevant, not only in JavaScript user interfaces (UIs), but in software distribution in general. The special relevance in JavaScript UIs is due to the increasing use of JavaScript in web applications, the legal risks associated with non-compliance, and the emphasis on software security and open-source adoption in recent government initiatives. Many companies are reluctant to take on any risks and are actively pursuing license compliance. Other companies do risk assessments to find out what problems related to license compliance or other open-source related topics they

should improve upon first. But, as shown in chapter 3.4.2, still far too many companies neglect license compliance in the context of JavaScript UIs. This is why it is important not only for managers but also for developers to understand obligations under open-source licenses and to take steps to ensure compliance, even in the context of JavaScript user interfaces.

3.6 About proper Attribution

After reading the first chapters of this thesis, *JavaScript user interface license compliance best practices*, one might conclude that attribution is only being done because certain laws must be followed. Attribution being the display of copyright notices and licenses in the form of third-party legal notices. Whilst the legal aspect is one of the main aspects that can motivate change in this sector to happen, it is not the only motor. The second motor is ethics. Giving credit, where credit is due and acknowledging the hard work numerous open-source developers have accomplished is what would be the *best thing to do*, but this is not yet enforced enough by current laws. The situation is also quite complicated since software is shared on a global scale, while copyright laws are still largely jurisdictional and not globally unified.

According to Tushnet (2007) attribution is currently largely incidental and customary, and authors who want attribution must depend on their economic leverage or norms of citation. When there is no contractual relationship between an author and a user, there is no way for an author to demand a separate attribution right. Tushnet also mentions that when the author of a work lacks economic leverage, they are unlikely to be able to retain attribution rights. This is often the case for individual creators when negotiating with large corporations. In many cases, the individual creator's work may be a work for hire, leaving them with no rights that copyright will recognize.

The situation is somewhat different for software licensing: Tushnet (2007) does mention software licensing in the context of computer programs and moral rights. They state that computer programs have so many utilitarian functions that moral rights would be incompatible with general social welfare. As a result, moral-rights theorists are less interested in bringing computer programs within the subject of moral rights. However, she also notes that open-source licenses generally involve attribution to contributors, suggesting that credit serves important functions for computer programmers, as it does for other types of authors.

This means software developers have an advantage. They do have a contract with the users of their software by licensing their work. If these licenses demand attribution, then any user of their software must do proper attribution. This means hiding the attribution notices behind a developer's console, as it is often done today (see chapter 3.4.2), is not acceptable (see chapter 5.3). Attribution notices need to be readily available to all users, not only to users with a computer science background that know how to operate a developer's console.

4 Objective Definition

The objective of this master thesis is the definition of a set of JavaScript user interface license compliance best practices. I will initially propose a set of guidelines that I will then extend to the optimal practices that one can adhere to, in order to ensure compliance with the FLOSS licenses that their software includes. As has been shown in prior chapters (especially 2.2.7, 3.1, 3.5, 3.6) everyone should be highly interested in doing or demanding so.

In order to ensure license-compliance when developing JavaScript applications, the following steps are essential and need to be carried out periodically whenever dependencies or the version of dependencies change.

- Generation of a Software Bill of Materials (SBOM) – Transitive closure!
- Analysis of every included component / package individually
 - Collection of Copyright Notices
 - Collection of Declared License(s)
 - Collection of Discovered License Texts or References
 - (Collection of Source Code)
 - (possibly more steps as required in the obligations of the individual licenses)
- Collection of findings, resolution of ambiguities, removal of false positives.
- Generation of a Third-party Legal Notice file – exact, neither sub- nor superset.
- Display of the Legal Notices in the Web context, easily accessible to the end-users. Not hidden. Not in the source code, where it is only accessible to developers.

In order to keep the scope of this thesis manageable, I will only focus on the generation of third-party legal notices and the Generation of the SBOM which is required for that. I will collect copyright notices, declared licenses, and license texts, but I will not collect source code or pursue other steps that compliance with the AGPLv3 or the GPLv3, for instance, would require.

5 Solution Design

Best practices are clearly defined activities in the scope of a workflow. Since the rule set I defined earlier foresees the creation of two artifacts, the SBOM and the third-party legal notices, I devise two workflows.

I assume the project does not introduce dependencies by any other way than by using npm with a *package.json* or *shrinkwrap.json* as it is called in more recent versions of npm. Such other means could be dynamically loading them on the client side or copy-pasting code into the project code files. It is also assumed that the project has a *package-lock.json* file and keeps it up to date.

Without further ado, I present the JavaScript user interface license compliance best practices.

5.1 JavaScript User Interface License Compliance Best Practices

(1) Create SBOM

The creation of the SBOM can be split into 4 smaller steps that in combination yield the SBOM. The creation of the dependency list, the identification of the licenses, copyrights, and other notices. It is worth noting that a minimal SBOM does not require any licensing or copyright information of the dependencies. However, if the goal is a third-party legal notice file, generating an enhanced SBOM is advisable, see chapter 2.5.

(a) Create dependency list

- Retrieve and parse the dependency list by gathering all “resolved” artifacts from the *package-lock.json*.

Due to the earlier assumptions, I know that the project’s entire dependency graph is stored in the *package-lock.json* file. This can be done facilitating “npm install” which downloads and stores the code of all packages in the *node_modules* folder.

When talking about JavaScript user interfaces that are distributed through the web, I can even go one step further. A website that is built and distributed means that only the dependencies and not the development dependencies in the *package-lock.json* need to be scanned, since only those are distributed along with the project’s own website code. That can be done by using `npm install --omit=dev`.

The *package-lock.json* file already gives us the full transitive closure. There is no need to do any more resolutions during further analysis.

(b) Identify license

- Identify declared license per dependency
 - Find the LICENSE file in the package, if not found, also check the README file

- Mark the detected license as a package wide license that is effective as long as the individual files don't say otherwise
- Match its contents with the open-source license list²⁴
- Store SPDX license identifier for exact match, store full license text otherwise
- Scan all files contained within the dependency. For each file:
 - Find comments that indicate a license (e.g., "Licensed under [...]", "See LICENSE file", "//MIT")
 - Evaluate if findings are correct – Human control mechanism
- Collect all findings in a list, grouped by package

(c) Identify and extract copyright notices

- Identify copyright notices per dependency
- Scan all files contained within the dependency. For each file:
 - Find Comments that indicate a copyright using regex matching (i.e., "©", "(c)", "(C)", "copyright")
 - Parse all files with regular expressions matching all possible ways to note a copyright
 - Evaluate if findings are correct – Human control mechanism
- Collect all findings in a list, grouped by package

(d) Identify and extract other notices

- Such can be, but are not exclusively:
 - Licenses appendices
 - Other notices
- Detect and Evaluate findings – Human control required

(2) Create third-party legal notice file

This workflow requires (1) Create SBOM. It is important to note that the third-party legal notices need to be precise. It is not permissible to incorporate information from a subset or superset of FLOSS packages other than those included in the software.

- List all dependencies in alphabetical order whilst using human-readable names
 - Add dependency name
 - For each identified license

²⁴ <https://opensource.org/licenses/>

- To retrieve the license text for the SPDX Identifier, either retrieve it from the license list or retrieve the exact license text that has been previously stored
- Append license text
 - List all copyright findings
 - Close with other notices
- Generate a table of contents and insert it at the start of the document.

5.2 Composition of the third-party Legal Notice

As shown in chapter 2.6, the third-party legal notice is by no means standardized yet. Which is why I have developed the following principles:

- The third-party legal notice must contain all copyright notices, declared licenses and license texts of the full transitive closure of the component that is being distributed.
- These license texts should not be summarized or aggregated, as some licenses when read carefully require an exact copy of the license text or portions of it to be retained.
- If technologically feasible a table of contents (TOC) needs to be added listing and linking all contained and redistributed FLOSS packages in human-readable form to facilitate ease of use.

For the scope of this master thesis I define the contents of a third-party legal notice file as follows:

```

<title>Third-Party Notices</title>

In case of questions, please contact EMAIL_ADDRESS_OF_PROVIDER
Table of contents

This software depends on external packages and source code.
All applicable license and copyright information is listed below.

<link>HUMAN_READABLE_COMPONENT_NAME_1</link>
  <spoiler>
    <link>SPDX License Identifier_1_1</link>
    <link>SPDX License Identifier_1_2</link>
    ...
  </spoiler>
<link>HUMAN_READABLE_COMPONENT_NAME_2</link>
...

<anchor><h1>HUMAN_READABLE_COMPONENT_NAME_1</h1></anchor>

<h2>SPDX License Identifier_1_1</h2>
LICENSE_TEXT_1(Component 1)

COPYRIGHT_NOTICE_1(Component 1, License 1)
COPYRIGHT_NOTICE_2(Component 1, License 1)
...

```

```
OTHER_NOTICES_1(Component 1, License 1)
...
<h2>SPDX License Identifier_1_2</h2>
LICENSE_TEXT_2(Component 1)

COPYRIGHT_NOTICE_3(Component 1, License 2)
...
<anchor><h1>HUMAN_READABLE_COMPONENT_NAME_2</h1></anchor>
...
THANKS!
...
EOF
```

Where HTML like tags indicate formatting and linking, respectively. All trailing “_n” indicate the nth numbered list element. As license and copyright information can differ from one software version to the next, different versions of the same component are treated as different components in this notice file. All other prose texts are verbatim texts.

It is also worth noting, that many common licenses have their own formatting as to where the copyright is placed within the license template. To avoid tearing apart the license templates, I recommend collecting the copyright notices in the place that is foreseen in most of the license templates.

When displayed in a website context, the third-party legal notices can and should be enhanced by the addition of a table of contents (TOC). This table should be a hyperlinked list of packages that links to the sections of the package where the package’s copyright notice, declared license and license text are shown. To further improve readability, I suggest making the contents of the table of content toggleable since one component can have countless licenses involved, which I indicated by using the spoiler tag. If the technology used to display the third-party legal notices does not allow for using toggleable headings in the TOC, the license identifiers are to be formatted bold instead of as h2 to remove them from the TOC.

“**THANKS!**” is meant to be replaced by a short and personal acknowledgement from the vendor of this software to the authors who provided their software under the conditions shown in this document. Any further refinements to the format should always aim to make third-party legal notices easier to read and understand for end users, and to enhance proper attribution.

5.3 Where to display Third-party Legal Notices to the User?

When considering the appropriate means of crediting authors, websites are not the first medium that comes to mind. In most cases, literature, such as this master thesis about JavaScript user interface license compliance best practices, follows a standardized approach for attribution. Attaching a source reference, as one does when using American Psychological Association (APA) styled citation²⁵, to every element of a website would likely appear cumbersome and visually unappealing. However, comparing a website to a document, the third-

25 <https://apastyle.apa.org/>

party legal notices on a website would be similar to the references found in this thesis. The main difference being that the references of this thesis represent only the first level of depth in a complex and nested dependency tree that led to its creation, while the third-party legal notice file includes information about every individual node in that dependency tree.

Again, comparing literature to a website, the correct place for the references in literature is the end of the document. In extension, for a website, that would mean the third-party legal notices should be displayed in the footer. I am well aware that displaying the entirety of the notice file at all time in the footer would neither be pleasant to look at nor performant, so to clarify this: A hyperlink titled “third-party legal notices” should be present in the footer of any website, and upon accessing this link the notices shall be directly presented to the user in a well readable manner and without leaving this website. Displaying the notices in the context of a different website would be comparable to sending the reader of this thesis to the store to buy a book where they can find the list of authors and works referenced in this thesis. Such a thing is unthinkable for any thesis, for example, but it is a common practice to handle third-party legal notices in the web context.

Whilst the overall attribution in a web context situation seems pretty dire, especially with the findings from chapter 3.6 in mind, appealing to our ethics, there are efforts being made. And in the context of software, the advantage of license agreements is always present. Section 5.d) of the GPLv3²⁶ for instance states, “if the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.” Which means that when a GPLv3 licensed software already does display the third-party legal notices in one way or another, any modification or addition also must display them. This indicates that attempts are made to improve the situation, but the impact of these attempts remains insignificant.

Websites frequently do not adhere to the same standardized approach for citing authors as literature does, however, valuable insights can be gained from the practice of incorporating references at the end of a document. Similar to references in literature, third-party legal notices on a website should be displayed in the footer, accessible through a hyperlink, to provide transparent and easily readable information about the dependencies involved in creating the website. And as to why to do so, I can refer to the lawsuits I analyzed that indicate an end-user right to file a lawsuit for breach of contract if no third-party legal notice is present.

26 <https://www.gnu.org/licenses/gpl-3.0.en.html>

5.4 Exemplary Application of Best Practices

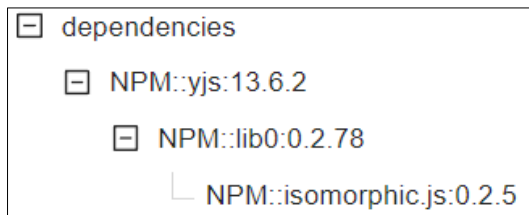


Figure 3: Dependency tree of the arbitrary project

To illustrate the best practices, I chose to use an arbitrary project that has only one dependency: `yjs`²⁷, as an example. Yjs is an open-source framework that enables developers to add real-time, collaborative features to their applications by providing a shared data structure that automatically synchronizes data across different devices and users.

The full dependency tree of the arbitrary project is shown in Figure 3. The package IDs shown in this figure are package IDs as used by the OSS Review Toolkit (ORT). I will use ORT in chapter 6 to demonstrate the application of the JavaScript user interface license compliance best practices at scale. An ORT package ID is of the format `<package manager>::<package name>:<package version>`.

1) Creation of the SBOM

1.a) Creation of the dependency list

- Parsing the whole `package-lock.json`
- Parsing the whole `node_modules` folder
- Creating a file list ordered by package

1.b) Identification of licenses

- `isomorphic.js`
 - LICENSE file found, marked as package wide main license
 - 100% match on MIT
 - Storing license Identifier (MIT) since it is a 100% exact match
 - Scanning remaining project files for file specific license information → No finds
- `lib0`
 - LICENSE file found, marked as package wide main license
 - 100% match on MIT
 - Storing license Identifier (MIT) since it is an exact match
 - Scanning remaining project files for file specific license information
 - Found references to CC0-1.0 and BSD-3-Clause licensed code → Stored the exact license text for the CC0 licensed code and the identifier for the 100% matching BSD-3-Clause licensed code
- `yjs`

²⁷ <https://github.com/yjs/yjs/tree/00ef472d68545cb260abd35c2de4b3b78719c9e4>

- LICENSE file found, marked as package wide main license
- 100% match on MIT
- Storing license identifier (MIT) since it is a 100% match
- Scanning remaining project files for file specific license information
- Found no license information saying otherwise

1.c) Identifying and extracting copyright notices

- isomorphic.js
 - Found copyright as part of MIT license:
 - Copyright (c) 2020 Kevin Jahns <kevin.jahns@protonmail.com>.
- lib0
 - Found copyright as part of MIT license:
 - Copyright (c) 2019 Kevin Jahns <kevin.jahns@protonmail.com>.
 - Found copyright of BSD-3-Clause licensed code parts.
 - Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura (this could not be found in the source code. It was missing. I needed to navigate to a link²⁸ that was referred to in the source-code to find the license information.
 - Found a public-domain license with authors that waived their copyright to the legal extent possible:
 - Written in 2016-2018 by David Blackman and Sebastiano Vigna (vigna@acm.org)
To the extent possible under law, the author has dedicated all copyright and related and neighboring rights to this software to the public domain worldwide. This software is distributed without any warranty.

See <<http://creativecommons.org/publicdomain/zero/1.0/>>.
- yjs
 - Found dual copyright in the LICENSE file:
 - Copyright (c) 2014
 - Kevin Jahns <kevin.jahns@rwth-aachen.de>.
 - Chair of Computer Science 5 (Databases & Information Systems), RWTH Aachen University, Germany

1.d) Collect all findings in a list, grouped by package

- The results of this step can be found within the following files in the *arbitrary-project-fossology-results* directory:

28 <http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/MT2002/CODES/MTARCOK/mt19937ar-cok.c>

- *SPDX2_arbitrary-project.zip_1687508536.spdx.rdf* provides info to each file
- *SPDX2CSV_arbitrary-project.zip_1686410140.csv* contains a file list sorted by package, referencing the items from the *.rdf* file

2) Creating third-party legal notice file

2.a) Listing all dependencies in alphabetical order, using human-readable names

- Add dependency name
- For each identified license:
 - Get license text of SPDX Identified license from the LicenseDB (only exact matches) or load previously stored exact license text otherwise
 - Append license text
- List all copyright findings
- Close with other notices

2.b) Generate a table of contents and insert it at the start of the document

- The result can be seen in *HandwrittenLegalNotice.md* or *HandwrittenMarkdownThird-party_Notices.pdf* located within the *arbitrary-project-tpln* directory and in Appendix A
-

5.5 Implementation of the Best Practices at Scale

Numerous tools have been developed to help ensure license compliance. I will provide an overview of the prevailing tools and select one of them to demonstrate my devised best practices in the context of a real project.

There are open-source license text, copyright scanners, and snippet matchers like ScanCode²⁹, FOSSology³⁰ and ScanOSS³¹. Open source container image analyzers, for example Tern³² and open source binary analysers, e.g., BAT / BANG³³. There are tools for open source workflow coordination, namely the OSS Review Toolkit (ORT)³⁴. And plenty of commercial scanners and matchers, among them Black Duck Hub (by Synopsys)³⁵, Flex Code Insight (by Flexera)³⁶, FOSSA (by FOSSA)³⁷, DejaCode (by nexB)³⁸ and Nexus IQ (by Sonatype)³⁹.

29 <https://github.com/nexB/scancode-toolkit>

30 <https://fossology.org>

31 <https://www.scanoss.com/>

32 <https://github.com/tern-tools/tern>

33 <https://binaryanalysis.org>

34 <https://github.com/oss-review-toolkit/ort>

35 <https://synopsys.com>

36 <https://flexera.com>

37 <https://fossa.io>

38 <https://dejacode.com>

39 <https://sonatype.com>

The ORT wiki also contains a list⁴⁰ of similar tools, which may be more up-to-date than this list. For additional details regarding their operation, I can refer to Ombredanne's (2020) column, where he provides a comprehensive overview of tools for software composition analysis.

According to Ombredanne, with scanning tools, you can detect

- structured information from package manifests and build scripts
- explicit license notices, license tags, license mentions, and license texts
- other provenance clues, including emails, uniform resource locators, and specific code constructs, such as programming language imports, include statements, namespaces, and code tree structures.

Scanning will not help you if there is no provenance or license information in the code you analyze (Ombredanne, 2020).

So, you can scan your codebase and the code of all components you import to find license texts, copyright notices and mentions. But what if a developer did copy some code from somewhere without including this information?

In that case, you would need to do matching. As the goal of matching is to find code that was borrowed from FOSS projects, based on the detection of code similarities. That means looking for duplicates, near duplicates, and clones (Ombredanne, 2020).

But, as I assumed earlier that the only way of inbound software is through the use of npm, I will not delve into further detail on matching here. There are several tools that support scanning, and these shall be the tools I focus on for the scope of this thesis.

The conundrum with all these open-source tools is that they require manual setup as a local instance or configuration, and thus a lot of tool-specific knowledge. This imposes an overhead for the average software developer that has often not yet gained experience in the usage of the respective tools. Even more so for the average website hosting user, that maybe does not have any software licensing knowledge at all.

Especially in the early stages of an emerging business, money, time, and knowledge are limited. The setup and usage of tools to ensure license compliance can be time and knowledge intensive. With the best practices that I developed for license compliance in JavaScript user interfaces, setting up and using such open-source tools can be easier. This is because of an understanding of the goals being pursued, what artifacts are required, and why these artifacts, namely the SBOM and the third-party legal notices, are required.

40 <https://github.com/oss-review-toolkit/ort/wiki/Similar-Tools>

6 Demonstration

For the demonstration of my devised best practices, I will make use of the Open-source-software Resource Toolkit (ORT) to generate a Software Bill of Materials, analyze the individual components and generate a third-party legal notice. The repository whose website I analyze is the repository of the QDAcity Project. QDAcity is a cloud-based web application for multiple researchers to collaboratively conduct Qualitative Data Analysis (QDA)⁴¹ in shared projects. Due to the goal of this master thesis, to find the best practices for license compliance in JavaScript user interfaces, I will restrict the analysis to the project files located in the `war` (web application resource) folder.

6.1 Setup of the OSS Review Toolkit

I first had to install ORT. This involved the installation of Docker⁴², the installation of the new Windows Subsystem for Linux (WSL2)⁴³, the enabling of virtualization⁴⁴, the cloning of the ORT repository and the build of the ORT image using the Docker buildkit⁴⁵.

Since there is no versioning for ORT yet, the “version” I used to generate the artifacts for this master thesis, is the main branch checked out at the commit with the following hash.

```
649902ca3e7311015fd95f744fd59c8c3ff34de3
```

This is the state of ORT at `Fri Feb 3 19:21:23 2023`.

I also installed `orthw`⁴⁶ which is a collection of helpful scripts bundled together to make using ORT more convenient.

In order to create the SBOM and the third-party legal notices, I went through the ORT workflow, which involves Analyzing, Scanning, Evaluating and Reporting on a project.

The analyze step of the ORT workflow reads all dependency information of a project. In the JavaScript context this usually involves reading and parsing the `package.json` and `package-lock.json`.

The scanning step launches and coordinates source code scanners, that scan the code of every dependency for any license or copyright findings. These findings are stored as intermediate results.

The evaluation step applies license governance rules to the intermediate results.

The report step generates various artifacts from the evaluated intermediate results. Some examples include exports of the SBOM as CycloneDX, SPDX files, or as a web app, to allow for easier manual review. It also allows exporting the third-party legal notices.

41 <https://qdacity.com/qda-help>

42 <https://docs.docker.com/get-docker/>

43 <https://learn.microsoft.com/de-de/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package>

44 <https://bce.berkeley.edu/enabling-virtualization-in-your-pc-bios.html>

45 <https://github.com/oss-review-toolkit/ort#build-using-docker>

46 <https://github.com/oss-review-toolkit/orthw/blob/main/README.md#installation>

6.2 Generation of the Software Bill of Materials

To generate the SBOM, an analysis followed by a report is sufficient.

I ran the first analysis with the following command, from the directory in which I checked out the QDacity repository.

```
docker run -v $PWD:/project ort --info analyze -i /project -f JSON -o /project/ort/analyzer
```

The first analysis concluded with the following output.

```
Exception in thread "main" java.lang.IllegalArgumentException: Unable to create the AnalyzerResult as it contains packages and projects with the same ids:
[[Package(id=Identifier(type=NPM, namespace=, name=babel-plugin-format-messages, version=1.0.0), purl=pkg:npm/babel-plugin-format-messages@1.0.0, cpe=null, authors=[], declaredLicenses=[], declaredLicensesProcessed=ProcessedDeclaredLicense(spdxEExpression=null, mapped={}, unmapped=[]), concludedLicense=null, description=, homepageUrl=, binaryArtifact=RemoteArtifact(url=, hash=Hash(value=, algorithm=)), sourceArtifact=RemoteArtifact(url=, hash=Hash(value=, algorithm=)), vcs=VcsInfo(type=, url=, revision=, path=), vcsProcessed=VcsInfo(type=, url=, revision=, path=), isMetadataOnly=false, isModified=false), Package(id=Identifier(type=NPM, namespace=, name=babel-plugin-format-messages, version=1.0.0), purl=pkg:npm/babel-plugin-format-messages@1.0.0, cpe=null, authors=[], declaredLicenses=[],
...
]]
```

Which told me that there are some ambiguities among the packages used. The packages with the following purls (package URLs⁴⁷, an aspiring standard for uniform naming of software packages) had conflicting IDs:

- pkg:npm/babel-plugin-format-messages@1.0.0
- pkg:npm/extract-messages-plugin@1.0.0
- pkg:npm/script-fetchpriority-plugin@1.0.0

This happened due to these packages being present within the package-lock.json whilst at the same time being present in the vendor directory.

I identified them as part of the development dependencies (devDependencies) which are not distributed. So, I opted to remove the devDependencies from the package.json, ran npm prune to update the package-lock.json and started the analysis again. I would at a later point learn that this crude approach and attempt at excluding development dependencies is not necessary, since ORT has its own way to define scope excludes.

The analysis concluded with the following message.

```
The analysis took 9m 42.327455451s.
Exception in thread "main" java.lang.IllegalArgumentException: The VcsInfo(type=Git, url=https://gitlab.com/profoss/qdacity.git, revision=, path=qdacity) of project 'NPM::qdacity:1.0.0' cannot be found in Repository(vcs=VcsInfo(type=, url=, revision=, path=), vcsProcessed=VcsInfo(type=, url=, revision=, path=), nestedRepositories={}, config=RepositoryConfiguration(analyzer=null, excludes=Excludes(paths=[], scopes=[]), resolutions=Resolutions(issues=[],
```

⁴⁷ <https://github.com/package-url/purl-spec/blob/master/README.rst#purl>

```
ruleViolations=[], vulnerabilities=[]), curations=Curations(packages=[],
licenseFindings=[]), packageConfigurations=[],
licenseChoices=LicenseChoices(repositoryLicenseChoices=[],
packageLicenseChoices=[])))
```

Upon brief inspection of the generated dependency tree, I found that this was a valid analyzer result and a valid SBOM and proceeded with the scan.

6.3 Analysis of every individual component

To obtain copyright notices, license texts, and license references, it is necessary to scan the source code of every dependency of the project using the ORT scanner.

```
docker run -v $PWD:/project -v $HOME/ort:/home/ort/ ort --info scan --
ort-file /project/ort/analyzer/analyzer-result.json -f JSON -o
/project/ort/scanner
```

The first run, however, yielded an exception.

```
Exception in thread "main" java.lang.IllegalArgumentException: The
VcsInfo(type=Git, url=https://gitlab.com/profoss/qdacity.git, revision=,
path=qdacity) of project 'NPM::qdacity:1.0.0' cannot be found in
Repository.
```

I attempted to resolve the issue initially by removing the dependency from the analyzer-result.json and using the reporter to generate some human-readable output:

```
docker run -v $PWD:/project -v $HOME/ort:/home/ort/ ort report --ort-
file /project/ort/analyzer/analyzer-result.json --output-dir
/project/ort/reporter --report-formats
AdocTemplate, CtrlXAutomation, CycloneDx, DocBookTemplate, EvaluatedModel, Excel,
FossId, GitLabLicenseModel, HtmlTemplate, ManPageTemplate, NoticeTemplate, Op
ossum, PdfTemplate, SpdxDocument, StaticHtml, WebApp, XhtmlTemplate
```

This yielded a valid scan report after 18 hours, but the main dependencies were missing. The scanner needed the correct Version Control System (VCS) information for the package. As it turned out, I initially analyzed the project from the wrong directory, in which one of the referenced directories was not present. I decided to run the analyzer for the top project level once more to take a look at the dependencies there. And indeed, that resolved the issue. After 16 hours, the following output was shown.

```
Writing scan result to '/project/ort/scanner/scan-result.json'.
02:21:23.929 [main] INFO org.ossreviewtoolkit.cli.commands.ScannerCommand
- Wrote ORT result to 'scan-result.json' (35.59 MiB) in 2.780216742s.
The scan took 16h 21m 41.419489522s.
```

Afterwards, the reporter was launched on the generated scan-result.json to inspect the outcome. The scan-result.json is a file that contains exact references to all findings of the scanners, i.e., a path and line numbers, but is not very human-readable in itself. The most human-readable output the reporter generates is the scan-report-web-app.html. It features a summary view, a table view and a tree view of the dependencies of the analyzed project,

which makes it much easier to understand the relationships between dependencies, and to find the origin of third-party dependencies. The third-party dependencies are software that has been introduced indirectly, embedded as a dependency of another software. Generating the same web report with `orthw` will create `webapp.html` by default instead, which is the same artifact, but named differently.

6.3.1 Narrowing down the analyzed Path

I closely followed the ORT documentation⁴⁸ on how to exclude certain paths from the analysis and – thanks to Andreas Kaufmann, CEO of QDAcity – figured out what parts of the project are actually part of the distributed UI.

The dependency tree that is of particular interest is the *qdcity:1.0.0* tree. A first look already shows us that the `devDependencies` (development dependencies) are still shown. These are of no interest for the analysis since they are not part of the distributed software, so I made use of `orthw` to move them out of scope.

```
orthw init
file:///Users/Me/Uni/Git-Masterarbeit/qdcity-martin-wagner/ort/scanner/
scan-result.json
```

That way, I could narrow down the initial scope to a much smaller actually analyzed scope, only including the relevant parts. The initial scope, for instance, also included the QDAcity development tools and their dependencies. To exclude these parts that are not included in the distribution, I made use of `path excludes`⁴⁹.

It should be noted that once all unrelated paths were excluded, there was no need to repeat the entire analysis and scanning procedure. Performance of any following scans could be optimized by scanning only the directories and scopes that were not excluded, if there was a need for future scanning or analysis steps. As during the initialization of `orthw` a “`.ort.yml`” configuration file was provided to exclude `devDependencies`, the `--skip-excluded` option can be used to avoid the download and scanning of that scope⁵⁰.

However, after this step, there were 100 ORT rule violations indicating that some human correction was necessary (see Figure 4). A rule violation in the context of ORT means, violation of rules that have been predefined during license governance. For this thesis I used the default rule-set that comes with any ORT installation. This default rule-set creates a rule violation for licenses that can not be identified as a match of any existing SPDX license, if patents or copy-left licenses are involved.

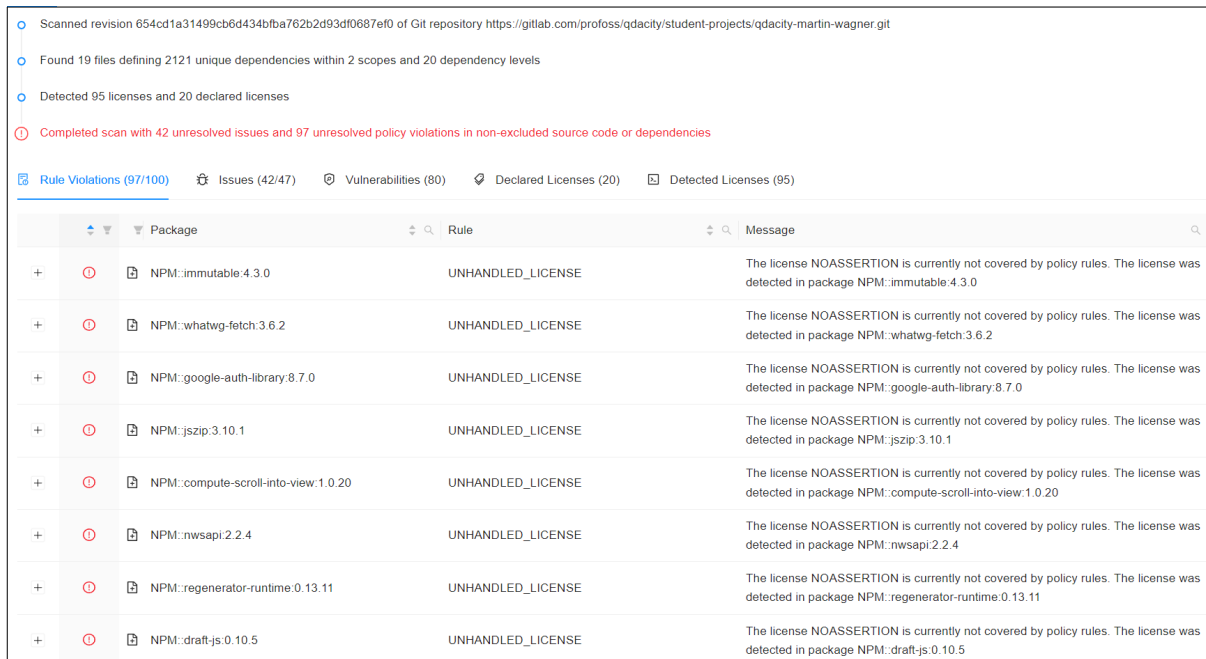
48 <https://github.com/oss-review-toolkit/ort/blob/main/docs/config-file-ort-yml.md#excluding-paths>

49 <https://github.com/oss-review-toolkit/ort/blob/main/model/src/main/kotlin/config/PathExcludeReason.kt>

50 <https://github.com/oss-review-toolkit/ort/blob/main/docs/getting-started.md#5-run-the-scanner>

6.4 Collection of Findings, resolution of Ambiguities, removal of false Positives

The next step was to review the automatically generated findings and resolve any ambiguities that have been escalated to me in the form of rule violations.



Scanned revision 654cd1a31499cb6d434bfa762b2d93df0687ef0 of Git repository https://gitlab.com/profoss/qdacity/student-projects/qdacity-martin-wagner.git

Found 19 files defining 2121 unique dependencies within 2 scopes and 20 dependency levels

Detected 95 licenses and 20 declared licenses

Completed scan with 42 unresolved issues and 97 unresolved policy violations in non-excluded source code or dependencies

Rule Violations (97/100) Issues (42/47) Vulnerabilities (80) Declared Licenses (20) Detected Licenses (95)

	Package	Rule	Message
+	NPM::immutable:4.3.0	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::immutable:4.3.0
+	NPM::whatwg-fetch:3.6.2	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::whatwg-fetch:3.6.2
+	NPM::google-auth-library:8.7.0	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::google-auth-library:8.7.0
+	NPM::jszip:3.10.1	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::jszip:3.10.1
+	NPM::compute-scroll-into-view:1.0.20	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::compute-scroll-into-view:1.0.20
+	NPM::nwsapi:2.2.4	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::nwsapi:2.2.4
+	NPM::regenerator-runtime:0.13.11	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::regenerator-runtime:0.13.11
+	NPM::draft-js:0.10.5	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::draft-js:0.10.5

Figure 4: Lots of NOASSERTION findings in the QDAcity web UI

To clear these rule violations, I looked into the source-code of the scanned packages as well as the distributed source and some websites in case they were referenced. To correct false metadata, as in incorrect scanner findings, or findings where no assertion could be drawn automatically, I made use of the `orthw pc-create-all` command, which leverages the *ort-config repository*. This repository already contains some curations, which are the ORT's way to correct false metadata of packages, contributed by other users and reviewed by the ORT team. By using this *ort-config repository* I was able to halve the number of licenses left to clear.

At this point in time 47 rule violations were remaining (see Figure 5).

Summary Table Tree

- Scanned revision 654cd1a31499cb6d434bfa762b2d93df0687ef0 of Git repository <https://gitlab.com/profoss/qdacity/student-projects/qdacity-martin-wagner.git>
- Found 19 files defining 2121 unique dependencies within 2 scopes and 20 dependency levels
- Detected 95 licenses and 20 declared licenses
- Completed scan with 42 unresolved issues and 44 unresolved policy violations in non-excluded source code or dependencies

Rule Violations (44/47) Issues (42/47) Vulnerabilities (80) Declared Licenses (20) Detected Licenses (95)

Package	Rule	Message
NPM:google-auth-library:8.7.0	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM:google-auth-library:8.7.0
NPM:jszip:3.10.1	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM:jszip:3.10.1
NPM:bignumber.js:9.1.1	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM:bignumber.js:9.1.1
NPM:googleapis-common:6.0.4	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM:googleapis-common:6.0.4
NPM:denque:1.5.1	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM:denque:1.5.1
NPM:styled-components:4.4.1	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM:styled-components:4.4.1
NPM:pdfjs-dist:2.5.207	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM:pdfjs-dist:2.5.207
NPM:leveldown:5.6.0	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM:leveldown:5.6.0

Figure 5: 47 rule violations remaining after usage of the ort-config repository

I then manually went through the remaining packages one by one and cleared any ambiguities.

6.5 No Assertion Findings

To clear NOASSERTION detections, I looked at the source code of the NOASSERTION findings, figured out what the license in this file is and added a license finding curation⁵¹ with the respective values for any finding.

```
license_finding_curations:
- path: "README.md"
  start_lines: "66"
  line_count: 1
  detected_license: "NOASSERTION"
  reason: "INCORRECT"
  comment: "Match on 'See [LICENSE]'."
  concluded_license: "Apache-2.0"
```

Most findings were easily resolvable. However, there were some that required more effort than others, I will present those in the following sections.

6.5.1 NPM::node-forge:1.3.1

The package NPM::node-forge:1.3.1 includes MIT licensed code with a custom addition (referenced by ORT as MIT-Addon). The obligations that come with the custom addition stating that “All redistributions must retain an intact copy of this copyright notice and disclaimer”⁵² are already implied by a pure MIT license, though.

51 <https://github.com/oss-review-toolkit/orthw/blob/main/docs/getting-started.md#correcting-invalid-or-missing-package-metadata>

52 <https://github.com/digitalbazaar/forge/blob/main/lib/jsbn.js>

des.js included a portion of code that had been altered and adapted from another author. However, the original license text is nowhere to be found⁵³. This is a license violation that was created by others, but due to QDAcity redistributing software that is not complying with their inbound software licenses, QDAcity could be held liable for this particular breach.

To resolve this ambiguity, I needed to find out under what type of license this modified code is and also try to find the original code with its license. Thankfully, the latter was fairly simple, since the original source code is hosted on the linked website⁵⁴ and the license text in the original source code is equal to the license text used in this file. The license text provided is a bare disclaimer and does not match exactly with any common license. I thus marked it with LicenseRef-scancode-warranty-disclaimer. On their website⁵⁵, Paul Tero also informs about the origin of their code. The JavaScript version of DES is based on a C version written and copyrighted by Eric Young as part of his SSL implementation, which can be found here⁵⁶ and here⁵⁷. Tracing down an exact version is almost impossible. Due to the time frame in which DES was created by Paul Tero, it is safe to assume that the license in effect when the derived work was created was the original SSLeay license.

Of especial interest is the last paragraph of this (almost BSD-4-clause) license:

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. I.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]⁵⁸

This means that also the modified and transcribed version of the DES algorithm would need to be licensed under this almost BSD-4-clause, copyleft license.

Since the OpenSSL toolkit at that time was double licensed, the OpenSSL license also applies. More recent versions of OpenSSL are now licensed under Apache 2.0. What all of this means, though, should be the concern of a law expert.

6.5.2 NPM::source-map:0.6.1 and NPM::source-map:0.5.7

Another mildly interesting find is the use of two different versions of the same packages in the same software at the same time (see Figure 6).

 NPM::source-map:0.6.1	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::source-map:0.6.1
 NPM::source-map:0.5.7	UNHANDLED_LICENSE	The license NOASSERTION is currently not covered by policy rules. The license was detected in package NPM::source-map:0.5.7

Figure 6: Different versions of the same software in one distribution can be found

53 <https://github.com/digitalbazaar/forge/blob/main/lib/des.js>

54 <http://www.tero.co.uk/des/>

55 <https://www.tero.co.uk/des/usage.php>

56 <https://github.com/openssl/openssl/blob/master/LICENSE.txt>

57 <https://www.openssl.org/source/license.html>

58 <https://www.openssl.org/source/license-openssl-ssleay.txt>

6.6 Copyleft Findings

There are a couple of copyleft (Or as classified by ORT as *COPYLEFT_IN_DEPENDENCY*) findings that are shown by ORT (see Figure 7)

	Package	Rule	Message
+	NPM::hellojs:1.20.0	COPYLEFT_IN_DEPENDENCY	The dependency 'NPM::hellojs:1.20.0' is licensed under the ScanCode 'copyleft' categorized license GPL-2.0-only.
+	NPM::hellojs:1.20.0	COPYLEFT_IN_DEPENDENCY	The dependency 'NPM::hellojs:1.20.0' is licensed under the ScanCode 'copyleft' categorized license GPL-1.0-or-later.
+	NPM::jszip:3.10.1	COPYLEFT_IN_DEPENDENCY	The dependency 'NPM::jszip:3.10.1' is licensed under the ScanCode 'copyleft' categorized license GPL-3.0-or-later.
+	NPM::jszip:3.10.1	COPYLEFT_IN_DEPENDENCY	The dependency 'NPM::jszip:3.10.1' is licensed under the ScanCode 'copyleft' categorized license GPL-3.0-only.

Figure 7: Copyleft findings in the QDAcity web UI

6.7 Inspection of License Choices and Patent Grants

I also found some packages that gave us a license choice⁵⁹. It is the ORT's way to mark a certain license as the one that is chosen for a certain dependency in this project's scope. In hindsight, this step may have been omitted since choosing a certain license does not necessarily have an effect on the SBOM or third-party legal notices. The implications of choosing one license of a dual licensed package are unclear. Does this mean that only the chosen license needs to be added to the third-party legal notices, or do both licenses and the note that a license can be chosen need to be added?

NPM::hellojs:1.20.0

This package includes some cases where the scanner found out that a license choice can be made. After investigating the source code at the time that it was referenced, it is clear that the conclusion is made correctly⁶⁰ and the MIT license is chosen.

NPM::scriptjs:2.5.9

The scan of this package showed a LGPL-2.0-or-later finding in `vendor/mootools.js`

The source claimed that it is MIT licensed while telling the reader about "inspiration":

```
- Class implementation inspired by
[Base.js](http://dean.edwards.name/weblog/2006/03/base/) Copyright (c)
2006 Dean Edwards, [GNU Lesser General Public License]
(http://opensource.org/licenses/lgpl-license.php)
- Some functionality inspired by [Prototype.js](http://prototypejs.org)
Copyright (c) 2005-2007 Sam Stephenson, [MIT License]
(http://opensource.org/licenses/mit-license.php)
credits:
- Element positioning based on the [qooxdoo](http://qooxdoo.org/) code and
smart browser fixes, [LGPL License]
(http://www.gnu.org/licenses/lgpl.html).
- Viewport dimensions based on [YUI](http://developer.yahoo.com/yui/)
```

59 <https://github.com/oss-review-toolkit/ort/blob/main/docs/config-file-ort.yml.md#license-choices>

60 <https://github.com/jquery/sizzle/blob/423f35af8bf43f3f07bb17284e21608f08372c22/LICENSE>

code, [BSD License](http://developer.yahoo.com/yui/license.html).⁶¹

And these lines have also been detected correctly by the scanner. All *MIT OR BSD-3-Clause OR GPL-1.0-or-later* and *MIT OR GPL-2.0-only* findings were traced to the three options that *jQuery* gave at that time. Again, if the work was “inspired” enough to be considered derivative work, the LGPL would apply to that portion of the code and come with its own obligations.

NPM::jszip:3.10.1

The first of the different findings for this package was *MIT OR GPL-3.0-only*, reflecting the declared dual license of *jszip*. I chose the MIT license. The scan also showed some *GPL-3.0-only* finds, which turned out to be the exact match in the LICENSE.markdown file. I corrected the discoveries in the package metadata to reflect *MIT OR GPL-3.0-only* and proceeded to include a license choice for this package.

Packages with additional patent grants

I also detected some packages that contained an additional patent grant. However, patent rights are out of scope for this master thesis and will thus only be briefly mentioned here. The ORT IDs of the packages in question are *NPM::draft-js:0.10.5*, *NPM::immutable:3.7.6*, *NPM::regenerator-runtime:0.10.5* and *NPM::react-intl:2.9.0*.


 NPM::mxgraph:4.2.2	FREE_RESTRICTED_IN_DEPENDENCY	The dependency 'NPM::mxgraph:4.2.2' is licensed under the ScanCode 'free-restricted' categorized license BSD-3-Clause-No-Nuclear-License. This requires approval.
 NPM::base64-arraybuffer:0.1.4	GENERIC_IN_DEPENDENCY	The dependency 'NPM::base64-arraybuffer:0.1.4' might contain a license which is unknown to the tooling. It was detected as LicenseRef-scancode-proprietary-license which is just a trigger, but not a real license. Please create a dedicated license identifier if the finding is valid.
 NPM::draft-js:0.10.5	PATENT_IN_DEPENDENCY	The dependency 'NPM::draft-js:0.10.5' is licensed under the ScanCode 'patent-license' categorized license LicenseRef-scancode-facebook-patent-rights-2. This requires approval.
 NPM::immutable:3.7.6	PATENT_IN_DEPENDENCY	The dependency 'NPM::immutable:3.7.6' is licensed under the ScanCode 'patent-license' categorized license LicenseRef-scancode-facebook-patent-rights-2. This requires approval.
 NPM::regenerator-runtime:0.10.5	PATENT_IN_DEPENDENCY	The dependency 'NPM::regenerator-runtime:0.10.5' is licensed under the ScanCode 'patent-license' categorized license LicenseRef-scancode-facebook-patent-rights-2. This requires approval.
 NPM::react-intl:2.9.0	PATENT_IN_DEPENDENCY	The dependency 'NPM::react-intl:2.9.0' is licensed under the ScanCode 'patent-license' categorized license LicenseRef-scancode-facebook-patent-rights-2. This requires approval.

Figure 8: Remaining rule violations after resolution of all ambiguities

After investigating all packages, there were 6 findings left (see Figure 8), that I was unable to resolve. The QDAcity developers will have to review and discuss these themselves.

6.8 Review of the detected Licenses

I then chose to investigate some of the more odd license findings. Those were not displayed as rule violations, since I did not configure the rules to treat these as violations. In my probes, I

61 <https://github.com/ded/script.js/blob/95ca7a60c75e266a492262e29dadb3ec2ea15a95/vendor/mootools.js#L4161>

found some false positives. Namely, *NPM::ieee754:1.2.1* where a “mplus” license was falsely detected.

I also took a look at *NPM::difflib:0.2.4* where a wrong version of the Python Software Foundation License Version 2 (PSFL) was referenced. I corrected this by creating a file in the *ort-config* directory with the actual license text and adding the created license to the *license-classifications.yml* file in said configuration directory. The declared license appears to be correct, although the only reference in the README.md I could find was “Ported from Python’s *difflib*⁶² module”, which is indeed licensed under PSFL-2.0.

I also took a look at *NPM::dnd-core:2.6.0* and found out that with the referenced version both BSD-3-Clause and MIT apply as already correctly discovered by the Scanner. All detections in the packages *NPM::encoding:0.1.13* and *NPM::heap:0.2.7* were correct. Furthermore, looking at the package *NPM::image-size:0.6.3*, the scanner did not know about the spec directory not being distributed since it is only used for testing and thus created a path-exclusion.

6.9 Generation of Third-party Legal Notices

Due to me setting up the *orthw* earlier, this step turned out to be very straightforward. After running *orthw report*, I was served with two third-party-legal-notice files. *NOTICE_BY_PACKAGE* and *NOTICE_SUMMARY*. The first file groups all findings by package, the latter only by license, which is not a good approach, as already mentioned in chapter 2.6. To inspect these files, please locate them within the *qdacity-ort-results* directory that has been distributed along with this master thesis.

This third-party legal notice does not completely fulfill the specification I created in chapter 5.2. It is mainly missing the table of contents and the *Thanks* section. However, it is a valid third-party legal notice and will surely deter anyone trying to “attack” QDAcity for some easy compensation by filing a lawsuit over breach of contract, after accessing their website.

In summary, I generated a SBOM with ORT by analyzing the QDAcity project. This SBOM was enriched with licensing information by scanning the source code of the entire transitive closure of dependencies. Evaluation and manual improvement of the license results helped refine the SBOM in its final form. The third-party legal notices could be derived from the enriched SBOM. A comprehensive understanding of ORT was essential to accomplish this task. The following slight deviations from my proposed best practices were observed. In ORT, the step of human verification and refinement is encapsulated in a separate process of repeatedly reporting, evaluating and clearing the SBOM. Scanning for licenses and copyright notices is combined into one step. The ORT reporter fails to produce comprehensive third-party notices because both the table of contents and the final acknowledgments section are missing. Other than that, the best practices can be mapped to the ORT workflow.

In order to avoid exceeding the scope of this thesis, it was decided to conclude the analysis here. It should be noted that there remains a set of tasks yet to be completed to ensure full JavaScript user interface license compliance.

62 <http://docs.python.org/library/difflib.html>

The subsequent tasks remain to be addressed:

- The clearance of the remaining potential issues I showed from chapter 6.5 onward.
- The display of these third-party legal notices in the web context of the application, easily accessible to the end-users and not hidden, and not only in the source code where it is only accessible to developers. Failure to do so can potentially have severe legal consequences (see also chapters 5.3 and 3.5).
- The continuous repetition of this process whenever any dependency or dependency-version changes and the continuous update of the SBOM and third-party legal notices.
- The manual, labor-intensive process of verifying every single scanner finding, to ensure 100% license compliance.
- The utilization of license governance practices to define how to handle copyleft licenses, for instance.

7 Comparison to FOSSology

With the intention to verify the generated SBOM and third-party legal notices, I decided to analyze the same project with another tool: FOSSology⁶³. FOSSology is an open-source software tool used for analyzing and managing the licenses and copyrights of FLOSS packages. This process involved the setup, scan and manual clearing of the scanner findings.

7.1 Setup of FOSSology

I opted for a setup with Docker.

- `docker pull fossology/fossology` pulls a prebuilt FOSSology instance
- I then created a FOSSology database and user in my PostgreSQL instance
- I ran the Docker image with the environment variables

```
docker run -d -p 8081:80 fossology/fossology
docker exec -it <container-id> /bin/bash
echo "FOSSOLOGY_DB_NAME=fossology" >>
/usr/local/etc/fossology/fossology.conf
echo "FOSSOLOGY_DB_HOST=host.docker.internal" >>
/usr/local/etc/fossology/fossology.conf
echo "FOSSOLOGY_DB_USER=fossy" >>
/usr/local/etc/fossology/fossology.conf
echo "FOSSOLOGY_DB_PASSWORD=fossy" >>
/usr/local/etc/fossology/fossology.conf
service fossology restart
```

63 <https://www.fossology.org/>

- And then navigated to <http://localhost:8081/repo/> and opened the FOSSology UI

7.2 File Upload, Scan and Clearance with FOSSology

I cloned the repository, ran `npm install`, to download and install all dependencies for the QDAcity UI, then proceeded to package the entire project as a `.zip` file and uploaded it. Figure 9 shows the selected settings. Upon completion, I was presented with a pleasant and somewhat intimidating overview (Figure 10).

4. Apply global decisions for current upload ⓘ

5. Ignore SCM files (Git, SVN, TFS) and files with particular Mimetype ⓘ

6. Visible only for active group ⓘ
 Visible for all groups ⓘ
 Make Public ⓘ

7. Select optional analysis:

- Copyright/Email/URL/Author Analysis
- ECC Analysis, scanning for text fragments potentially relevant for export control
- IPRA Analysis, scanning for text fragments potentially relevant for patent issues
- Keyword Analysis
- MIME-type Analysis (Determine mimetype of every file. Not needed for licenses or buckets)
- Monk License Analysis, scanning for licenses performing a text comparison
- Nomos License Analysis, scanning for licenses using regular expressions
- Ojo License Analysis, scanning for licenses using SPDX-License-Identifier
- Package Analysis (Parse package headers)
- REUSE.Software Analysis (forces *Ojo License Analysis*)
- SCANOSS Toolkit
- Software Heritage Analysis

8. Automatic Concluded License Decider ⓘ, based on

- Scanners matches if all Nomos findings are within the Monk findings
- Scanners matches if Ojo or REUSE.Software findings are no contradiction with other findings
- Bulk phrases from reused packages
- New scanner results, i.e., decisions were marked as work in progress if new scanner finds additional licenses
- False Positive Copyright Deactivation (**experimental**) ⓘ
- False Positive Copyright Deactivation and clutter removal (**experimental**) ⓘ

9. ScanCode Toolkit ⓘ, scan for

- License ⓘ
- Copyright ⓘ
- Email ⓘ
- URL ⓘ

Figure 9: FOSSology settings

Thankfully, to fulfill the task of ensuring license compliance of the QDAcity JavaScript UI, I only need to check the `war` directory, since this is where the web application is at home. After running `npm install`, this directory already contains the full transitive closure of dependencies that are distributed along with the web UI.

After the first scan, I quickly realized that using `npm install` to install all dependencies might not be sufficient to get the full dependency tree. When inspecting the dependency tree as resolved in the ORT analyze step, I find many packages that are not found automatically in

the package uploaded to my local FOSSology instance. After a brief look into the files I uploaded, I found that running `npm install` in the main directory, but not the war directory, was the reason for the missing dependencies.

	-- filter for scan results --	-- filter for edited results --	<input type="checkbox"/> Open		Edit Decisions
Files	Scanner Results (N: nomos, M: monk, Nk: ninka, I: reportImport, O: ojo, Sc: scancode, Sp: spasht, Rs: reso, So: scanoss)	Edited Results	Clearing Status	Cleared / Open / Total	Actions <input type="checkbox"/>
node_modules	0BSD, AFL-2.1, AGPL-3.0-only, AGPL-3.0-or-later, Apache-2.0, BSD, BSD-2-Clause, BSD-3-Clause, BSD-3-Clause-No-Military-License, BSD-possibility, BSD-style, CC-BY, CC-BY-3.0, CC-BY-4.0, CCO-1.0, Dual-license, EPL-2.0, GPL, GPL-1.0-or-later, GPL-2.0-only, GPL-3.0-only, GPL-3.0-or-later, ISC, ISC-possibility, LGPL, LGPL-2.0-or-later, LGPL-2.1-only, LGPL-2.1-or-later, LGPL-3.0-only, LicenseRef-scancode-facebook-patent-rights-2, LicenseRef-scancode-mit-addition, LicenseRef-scancode-public-domain, LicenseRef-scancode-unknown-license-reference, LicenseRef-scancode-warranty-disclaimer, Linux-OpenIB, MIT, MIT-0, MIT-possibility, MIT-style, MPL-2.0, No_license_found, OFL-1.1, OLDAP-2.8, PSF-2.0, Public-domain, Python-2.0, See-URL, See-doc.OTHER, See-file, UnclassifiedLicense, Unlicense, WTFPL, WebM, X11, X11-distribute-modifications-variant, X11-possibility, Zlib	MIT	●	43 / 24423 / 37491	[Tag][Edit][Bulk]

Figure 10: Result overview shown after scanning the QDAcity war node_modules directory

The use of SCANOSS yields many false positives, that all need to be cleared manually. After consulting the FOSSology documentation, I could not figure out why that would be. So, I simply ran another scan without using SCANOSS. Many findings of the word “require” are classified as GPL findings. So, using FOSSology and manually clearing a project will put you on the absolute safe side. This comes at the cost of a tiring process of manual clearing every single finding that the aggressive scanners display and escalate to you. In some cases, this process might even introduce errors, if the user doing the clearing work makes a mistake.

Since I knew I only needed to focus on distributed parts of the packages, I chose to give it one more try and used `npm install --omit=dev` to install the dependencies used in production only.

After clearing a couple of packages by hand, it quickly became apparent to me that this task is out of scope for this master thesis. The scanner showed over 30000 license findings, although about 20000 of them were from the @fontawesome packages.

7.3 Comparison of ORT and FOSSology Results and their Limitations

Finally the licenses found by FOSSology have been compared to the licenses that the ORT scanner found. And indeed, upon comparing the list of detected licenses, ORT actually found more different licenses than FOSSology, while covering all the licenses found by FOSSology apart from different naming. E.g., a license finding with the identifier *public-domain* in FOS-

Sology would be found under *LicenseRef-scancode-public-domain-disclaimer* in the ORT results.

The implications of the *transitive closure* become evident when looking at some numbers from the analysis of the QDAcity project. The QDAcity UI declares 47 dependencies. The actual number of packages included is 302. And since the QDAcity UI is redistributing all of these packages, it has to be in compliance with every license from all of these 302 packages.

When utilizing ORT to establish license compliance and generate third-party legal notices for your project, there are still instances where human corrections and input are required. The vast majority of the work, however, is automated. Some nuances might not be detected by the automatic scanners, which might lead to small mistakes in the generated artifacts. Still, using the ORT in its default configuration will give any possible “attacker” that would like to sue you over a breach of contract a much smaller target.

For absolute safety, you should consider using a tool like FOSSology, which escalates everything that the scanners detect to the user, and requires manual confirmation of every single finding. If one has the time and expertise to pursue this approach, it would undoubtedly be the prudent course of action.

Interesting findings

To conclude the demonstration, I would like to show some of the more peculiar findings.

One of the more advanced LICENSE files I found was the *loadash* license file. It includes the following disclaimer:

```
Files located in the node_modules and vendor directories are externally maintained libraries used by this software which have their own licenses; we recommend you read them, as their terms may differ from the terms above.
```

Which is a nice way to inform users that there might be more to it than meets the eye.

Another peculiar find I made is that apparently everything goes for package descriptions.

```
Id: NPM::indexof:0.0.1
Package URL: pkg:npm/indexof@0.0.1
Description: Microsoft sucks
Source Artifact: https://registry.npmjs.org/indexof/-/indexof-0.0.1.tgz
```

Figure 11: The index of Microsoft

The third-party legal notice file generated by the ORT does not reflect license decisions as of now. I am aware that this is by no means necessary as the NOTICE file is meant to be distributed along with the software, to fulfill any acknowledgment or attribution requirements. Still, it might be necessary to be reflected somewhere. And how is this treated legally? If a vendor of a component picks a license, then what applies to a third party reusing this component? Will the third party have to oblige to the chosen license, or can a third party make a choice of their own again? These are some remaining questions that a lawyer could possibly answer.

8 Conclusion

The main research questions of this work can now be answered:

1. *What are the best practices for license compliance in the context of JavaScript user interfaces?*

The practices and considerations presented in this thesis provide a possible answer to this question.

2. *How can these best practices be applied at scale?*

The demonstration performed as part of this work presents a way to perform JavaScript user interface license compliance best practices at scale.

To answer these research questions in a scientifically well-founded manner, basic concepts on the subject of license compliance were explained in chapter 2. In chapter 3, the extent of the problem of lacking proper attribution and license compliance in the context of JavaScript user interfaces was shown. As a result, it was determined that everyone – software developers, managers, and end users – is responsible for ensuring that license obligations are met. It was also shown that most license obligations come into effect, as they cross the third-party boundary. The ever-increasing popularity of JavaScript in combination with the ever-growing amount of FLOSS being used will only escalate the situation further if no measures are taken to improve the situation. Recent legal cases are steering in a direction where every JavaScript user interface will be legally required to be in compliance with all involved licenses. Should license infringements be considered a breach of contract in the future, every end user can sue the vendor of a website over breach of contract. And even in the absence of legal actions, one should already be doing license compliance correctly, as our ethics dictate that everyone does so.

The rule set I devised in chapter 4 of this master thesis was extended and refined into a set of best practices that take place within the framework of two main workflows in chapter 5. The first workflow being the creation of an SBOM followed by a second workflow composing the third-party legal notices.

As shown in chapter 6 of this master thesis, it is feasible to create a SBOM for a project and generate a notice file using FLOSS like ORT. But it requires a lot of knowledge about how ORT works and its configuration. This information and the use of these tools is not easily accessible to the average website hosting person, and the workflow is still quite cumbersome. It involves copying and pasting information like URLs and in some cases license texts manually, which is error-prone. At the same time using tools like ORT is a valid best effort when tackling license compliance of JavaScript user interfaces in the context of vast projects. The quality of the resulting artifacts, being the SBOM and the third-party legal notice file, is good, but not quite perfect yet, as shown in chapter 7. Other tools have different approaches. FOS-Sology for instance, utilizes a more manual labor heavy approach. This can, depending on the

skills of the human performing the license clearing, improve or diminish the quality of the results compared to the mostly automatically generated results of ORT.

The significance of rigorous license compliance in JavaScript User Interfaces, pursuing the best practices elaborated in this thesis, cannot be overstated. The continuous creation of the SBOM and generation of the third-party legal notices, facilitated by tools like ORT, needs to be required to ensure both ethical and legal obligations are met at scale. Should there be apprehensions about managing the setup and configuration of tools like ORT or reluctance to make considerable financial investments to ensure license compliance of your website, I am currently developing a solution: SCATool. This web application aims to make the license compliance process as user-friendly and autonomous as possible. It is of great importance to remember our collective responsibility to ensure appropriate attribution and license compliance as we navigate through the rapidly evolving FLOSS landscape influencing our digital era.

Appendix A Arbitrary Project Third-party Legal Notices

Third-party Notices

In case of questions, please contact `EMAIL_ADDRESS_OF_PROVIDER`

Table of contents

- [Package: isomorphic.js:0.2.5](#)
 - [MIT](#)
- [Package: lib0:0.2.78](#)
 - [CC0-1.0](#)
 - [MIT](#)
 - [BSD-3-Clause](#)
- [Package: yjs:13.6.2](#)
 - [MIT](#)

This software depends on external packages and source code. All applicable license information is listed below.

Package: isomorphic.js:0.2.5

The following copyrights and licenses were found in the source code of this package:

MIT

Copyright (c) 2020 Kevin Jahns kevin.jahns@protonmail.com

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, subject to the following conditions:

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Package: lib0:0.2.78

The following copyrights and licenses were found in the source code of this package:

CC0-1.0

Written in 2016-2018 by David Blackman and Sebastiano Vigna (vigna@acm.org)

To the extent possible under law, the author has dedicated all copyright and related and neighboring rights to this software to the public domain worldwide. This software

is distributed without any warranty.

See <http://creativecommons.org/publicdomain/zero/1.0/>.

MIT

(c) Kevin Jahns Copyright (c) 2019 Kevin Jahns kevin.jahns@protonmail.com

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, subject to the following conditions:

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

BSD-3-Clause

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Package: yjs:13.6.2

The following copyrights and licenses were found in the source code of this package:

MIT

Copyright (c) 2014

- Kevin Jahns kevin.jahns@rwth-aachen.de.
- Chair of Computer Science 5 (Databases & Information Systems), RWTH Aachen University, Germany

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, subject to the following conditions:

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

THANKS!

References

- Ballhausen, M. (2019). Free and Open Source Software Licenses Explained. *Computer*, 52(6), 82–86.
<https://doi.org/10.1109/MC.2019.2907766>
- Deshpande, A., & Riehle, D. (2008). The Total Growth of Open Source. *OSS*, 275, 197–209.
- Fogel, K. (2005). *Producing open source software: How to run a successful free software project*. O'Reilly Media, Inc.
- Free Software Foundation. (2019, July 30). *What is Free Software? - GNU Project*.
<https://www.gnu.org/philosophy/free-sw.en.html>
- GitHub. (2022). *Octoverse 2022: The state of open source*. The State of the Octoverse.
<https://octoverse.github.com/>
- Haddad, I. (2016). Open Source Compliance in the Enterprise. *The Linux Foundation. Disponible Em*.
- Harutyunyan, N., & Riehle, D. (2021). Getting Started with Corporate Open Source Governance: A Case Study Evaluation of Industry Best Practices. *Proceedings of the 54th Hawaii International Conference on System Sciences*, 6263.
- International Organization for Standardization. (2015). *Software identification tag (ISO/IEC No. 19770-2:2015)*.
<https://www.iso.org/standard/65666.html>
- International Organization for Standardization. (2020). *OpenChain Specification (ISO/IEC 5230:2020)*.
<https://www.openchainproject.org/license-compliance>
- International Organization for Standardization. (2021). *SPDX® Specification V2.2.1 (ISO/IEC 5962:2021)*.
<https://www.iso.org/standard/81870.html>
- Laurent, A. M. S. (2004). *Understanding open source and free software licensing: Guide to navigating licensing issues in existing & new software*. O'Reilly Media, Inc.
- Matt Compton. (2015, July 8). Open-Source Licenses and Android. *Big Nerd Ranch*.
<https://bignerdranch.com/blog/open-source-licenses-and-android/>
- Meeker, H. (2017). *Open (Source) for Business: A Practical Guide to Open Source* (2nd ed.). CreateSpace Independent Publishing Platform.
- National Institute of Standards and Technology. (2021). Executive Order 14028, Improving the Nation's Cybersecurity. *NIST*. <https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity>

- National Institute of Standards and Technology. (2022). Software Security in Supply Chains: Software Bill of Materials (SBOM). *NIST*. <https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity/software-security-supply-chains-software-1>
- Ombredanne, P. (2020). Free and Open Source Software License Compliance: Tools for Software Composition Analysis. *Computer*, 53(10), 105–109. <https://doi.org/10.1109/MC.2020.3011082>
- Open Source Initiative. (2007, March 22). *The Open Source Definition*. <https://opensource.org/osd>
- Open Source Initiative. (2007, October 21). *What is “free software” and is it the same as “open source”?* <https://opensource.org/faq/#free-software>
- Phipps, S., & Zacchioli, S. (2020). Continuous Open Source License Compliance. *Computer*, 53(12), 115–119. <https://doi.org/10.1109/MC.2020.3024403>
- Riehle, D., & Harutyunyan, N. (2019). Open-source license compliance in software supply chains. In *Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability* (pp. 83–95). Springer Singapore. <https://doi.org/10.1007/978-981-13-7099-1>
- Riehle, D., & Lempetzeder, B. (2014). *Erfolgsmethoden der Open-Source-Governance und Compliance* [Technical Reports]. Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science.
- Rosen, L. E. (2005). *Open source licensing: Software freedom and intellectual property law*. Prentice Hall PTR.
- Ruffin, C., & Ebert, C. (2004). Using open source software in product development: A primer. *IEEE Software*, 21(1), 82–86. <https://doi.org/10.1109/MS.2004.1259227>
- Software Freedom Conservancy. (2022, May 16). *Software Freedom Conservancy right-to-repair lawsuit against California TV manufacturer Vizio, Inc. Remanded to California State Court*. <https://sfconservancy.org/news/2022/may/16/vizio-remand-win/>
- Software Freedom Conservancy, Inc. V. Vizio, Inc. Et al, 8:21-cv-01943-JLS-KES (C.D. Cal. May 13, 2022). <https://www.courtlistener.com/docket/61578720/30/software-freedom-conservancy-inc-v-vizio-inc/>
- Stack Overflow. (2022). *Stack Overflow Developer Survey 2022*. https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022
- Synopsys Editorial Team. (2022, July 13). *Top open source licenses and legal risk for developers*. Application Security Blog. <https://www.synopsys.com/blogs/software-security/top-open-source-licenses/>

- The Apache Software Foundation. (2004, January). *Apache License, Version 2.0*.
<https://www.apache.org/licenses/LICENSE-2.0.html#redistribution>
- The United States Department of Commerce. (2021). *The Minimum Elements For a Software Bill of Materials*.
https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
- The White House. (2021, May 12). *Executive Order on Improving the Nation's Cybersecurity*. The White House.
<https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/>
- Top Websites Ranking in Deutschland in Januar 2023*. (n.d.). Similarweb. Retrieved February 18, 2023, from
<https://www.similarweb.com/de/top-websites/germany/>
- Tushnet, R. (2007). Naming Rights: Attribution and Law. *Utah L. Rev.*, 781–814.
- VandeVondele, J. (2022, November 18). *ChessBase GmbH and the Stockfish team reach an agreement and end their legal dispute—Stockfish—Open Source Chess Engine*.
<https://stockfishchess.org/blog/2022/chessbase-stockfish-agreement/>
- Wagner, J. (2022). The 2022 Web Almanac: JavaScript. In *The 2022 Web Almanac* (Vol. 4, Issue 2). HTTP Archive. <https://almanac.httparchive.org/en/2022/javascript>
- Zacchiroli, S. (2022). A large-scale dataset of (open source) license text variants. *Proceedings of the 19th International Conference on Mining Software Repositories*, 757–761.
<https://doi.org/10.1145/3524842.3528491>
- Zittrain, J. (2004). Normative Principles for Evaluating Free and Proprietary Software. *The University of Chicago Law Review*, 71(1), 265–287.