DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Information Systems

Design of a System to Support Open Source Software License Compliance Checking in Large Enterprises

Navina Lang

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Information Systems

Design of a System to Support Open Source Software License Compliance Checking in Large Enterprises

Design eines Systems zur Unterstützung von Open Source Software Lizenz Compliance Checks in großen Unternehmen

Author:	Navina Lang
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Nektarios Machner
Advisor:	Dr. Ralf S. Engelschall
Submission Date:	10th September 2020

I confirm that this bachelor's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 10th September 2020 Navina Lang

Acknowledgments

Many thanks to Prof. Dr. Matthes and my supervisor Nektarios Machner for the support on the university side.

Special thanks to Dr. Ralf S. Engelschall for the excellent support in all matters, the high degree of compromise regarding the choice of topics, and the motivating words. These had helped me, especially in times when things were not going so well. By working together, I not only learned a lot, but I also enjoyed it a lot. I could not have imagined better support.

Next, I would like to thank the development team: Zoltan Ruzman, Moritz Hüther, Maximilian Marsch, Simon Bortnik, and Thomas Riexinger. Once again, special thanks go to Zoltan, who always supported me with questions and explained many things to me technically.

I would also like to thank Cornelia Seraphin and Pirjo Friedrich for their support regarding requirements engineering and the creation of the mockups. Many thanks to Erwin Wacha for introducing me the tool Adobe XD for the creation of mockups. Also, many thanks to Mark Lubkowitz for providing information about current Open Source Software (OSS) topics and reports.

Furthermore, I would like to thank my interview partners and the entire XT team for the pleasant cooperation and moral support.

Also, I would like to thank Sascha Nägele for supporting me and helping me to find a supervisor for the bachelor thesis on the university side.

I also would like to thank Julian Feldmeier for proofreading my bachelor thesis and giving me advices for improvements.

I am also very thankful to Margaret and Rodolfo Pérez for proofreading my bachelor thesis.

Finally, I would like to thank my mother Marina Lang for always supporting me and making all my desires possible. Also, thanks for proofreading, although there was no interest in the topic.

Abstract

The massive adoption of Open Source Software presents companies with new challenges. When selecting components, software architects have to consider not only quality but also the obligations of the license conditions. When using Open Source Software components, many transitive dependencies arise, which can also cause problems and therefore, must be checked. This bachelor thesis deals with the challenges of architects and presents a solution approach for supporting the license compliance process. Therefore a design for an application is presented, which is adapted to the requirements of the end-users. The approach focuses primarily on the challenges of large enterprises where defined processes are essential. The application also helps to facilitate the cooperation of the different disciplines. An architect is not educated on legal texts and therefore needs help in checking licenses. Therefore he needs evaluation in a language he understands. A legal expert, on the other hand, is not familiar with the technical construction of software components and therefore has difficulties in checking the compliance of the conditions.

Keywords: Open Source Software (OSS), Business Information Systems (BIS), Commercial Software Industry, Large Enterprises, and Decentralized Organizations.

Contents

1 Introduction

1.1 Motivation

The license compliance process is an interdisciplinary problem. On the one hand, it requires the know-how of legal experts and, on the other hand, the know-how of architects. However, these two specialists are not educated on the other discipline, which leads to problems in practice. Nowadays, the development of information systems is no longer feasible without the use of Open Source Software (OSS), since otherwise, companies are no longer competitive. An information system usually contains about 100-1000 components. However, the architect does not know many of these components as he is not aware of using them. The components contain many transitive dependencies and are deeply nested. However, uncertainty does not protect against punishment, and therefore it is necessary to check all transitive dependencies and comply with their license terms. Creating a blacklist of components that are not allowed to be used is not an option because the restrictions depend on the type of use. If, for example, the General Public License (GPL) is blacklisted, Unix servers are no longer allowed to be used for software development since they are based on a GNU/Linux-distribution. Both the Linux Kernel and several other tools are licensed under a GPL license. However, the concrete type of use is a problem, not the license itself. Using a GPL-licensed tool during software development is legally harmless. However, integrating a GPL-licensed library into a commercial product as an integral part is almost impossible. [1]

1.2 Research Questions

- What are the challenges of using OSS in terms of legal compliance?
- How can we improve the OSS license compliance legal process?
- What are the benefits and limitations of the proposed solution?

1.3 Research Approach

Design Research is an approach for Information System Research developed by Hevner et. al in 2004. He refers to two paradigms to characterize research in information systems discipline: behavioral science and design science.

Behavioral science has its roots in natural science research methods and addresses research by developing and verifying "theories that explain or predict human or organizational

behavior"[2]. It is important to understand the problem domain and the solution based on prior theories and existing design knowledge. The method of this thesis builds on an already existing method, which needs to be verified and improved to comply with the behavioral science.[2]

On the other hand, design science has its roots in engineering and addresses research by extending boundaries of human and organizational capabilities to create new and innovative artifacts. It is inherently a problem-solving process. Seven principles are derived from the principle of design science "knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact"[2].

Guideline 1: Design of an Artifact

"Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation."[2] This thesis provides a method to deal with license compliance in large enterprises and a UML data model based on this method. Therefore, this thesis adheres to the first guideline.

Guideline 2: Problem Relevance

"The objective of design-science research is to develop technology-based solutions to important and relevant business problems."[2] The artifacts solves the challenge architects have when using OSS components. Therefore, architects must comply with the license agreement, which is challenging since they are not educated in legal issues.

Guideline 3: Design Evaluation

"The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods."[2] Through the industrial support and the evaluation by the end-users a continuous evaluation is guaranteed.

Guideline 4: Research Contributions

"Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies."[2] The solution is based on a previously used method, verifying the methodology, and improving the method.

Guideline 5: Research Rigor

"Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact."[2] Rigor is applied by building upon an already tested method that needs improvement and evaluating relevant stakeholders.

Guideline 6: Design as a Search Process

"The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment."[2] The method is improved continuously through research and license analysis. Furthermore, the data model is continuously improved through several iterations and new findings.

Guideline 7: Communication of Research

"Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences."[2] The method and the resulting data model is technically realizable and intends to support the architect in the license compliance process. To comply with the license terms is also in the management's interest since violation can result in high costs and damage the company's reputation.

2 Foundations

2.1 Enterprise Terms

2.1.1 Large Enterprises

Large companies are mainly characterized by the number of employees, which exceeds 500. Besides, large companies usually have several locations, and the employees are distributed decentrally. In large companies, defined processes are essential since they are unlikely to know everybody and talk about the concerns when meeting in the coffee kitchen.

2.1.2 Agile Software Engineering

Scrum is an agile management framework. It defines self-organized, cross-functional teams, which work decentralized to fulfill requirements. There are no defined compliance processes, which makes it even more challenging to control the license compliance. Furthermore, it is challenging to keep the overview.

2.2 Architecture Terms

2.2.1 Architecture

"Inherent static and dynamic structure of a Subject which comprise Elements, the visible Behaviour of Elements and Relationships between Elements." [3]

2.2.2 Component

"Definition of a Component (of a Larger Whole): a know-how encapsulating, potentially reusable and substitutable unit of hierarchical composition with explicit context dependencies, which hides the complexity of its optional behavior and state realization behind small contractually sepcifed interfaces, defines its added value in terms of provided and consumed interfaces and optionally belongs to zero or more categories of similar units." [3]

2.2.3 Plug-in

A plug-in is an instrument that actively controls the application by connecting via Service Provider Interfaces (SPI). [3]

2.2.4 Framework

A framework is a semi-finished software system consisting of a large number of coordinated software components from which an adapted software system can be created with relatively little effort. [4]

2.2.5 Library

A library is a passive, reusable software component that offers functionality to the application via Application Programming Interfaces (API). [3][5]

2.2.6 Tool

A software tool is defined as "a computer program used in the development, testing, analysis, or maintenance of a program or its documentation" and as a "software product providing automatic support for software life-cycle tasks".[6]

2.2.7 Program

A program is a "syntactic unit that conforms to the rules of a particular programming language and that is composed of declarations and statements or instructions needed for a certain function, task, or problem solution"[6]. This definition defines the point of view of the source code. From the user point of view, a program is an executable software unit.

2.2.8 Operating System

An operating system is defined as a "program that acts as an intermediary between a user of a computer and the computer hardware"[7]. "Operating system goals"[7] are: "Execute user programs and make solving user problems easier", "Make the computer system convenient to use"[7] and "Use the computer hardware in an efficient manner"[7].

2.2.9 Technology Platform

A Technology Platform consists of a language, an optional run-time environment, and a standard library. [3]

2.2.10 Technology Stack

A Technology Stack extends a Technology Platform by frameworks and libraries. [3]

2.3 Open Source Software Terms

2.3.1 Open Source Software

Open Source Software is software that is licensed under an Open Source License. All Open Source Licenses comply with the Open Source Definition, which states, i.e., that the software may be further distributed and modified in source code and, in particular, that there may be no discrimination of persons, groups, or purposes. [3]

2.3.2 Free Software

"'Free software' means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, 'free software' is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer'. We sometimes call it 'libre software', borrowing the French or Spanish word for 'free' as in freedom, to show we do not mean the software is gratis."[8]

2.3.3 Free/Libre and Open Source Software (FOSS/FLOSS)

Free/Libre and Open Source Software is software that can be defined by Open Source Software as well as Free Software.

2.4 Open Source Software License Terms

2.4.1 Obligations

Obligations indicate conditions that must be fulfilled according to the license agreement.

2.4.2 Use Types

A Use Type indicates how the architect / developer uses a component.

2.4.3 License

Each OSS component is under a specific license. A license usually is a US-American formulated text that restricts the use of the component or imposes certain conditions. These conditions can be completely harmless, but they can also be an absolute rejection criterion for use with commercial software.

2.4.4 Prose Text

The prose text represents the text excerpts of respective licenses that contain relevant information regarding the obligations.

2.4.5 Copyright

To understand most OSS licenses, it is helpful to look at some key differences between the Anglo-American "Copyright" and the continental European, here before all German, "copyright" before eyes:

German copyright law is designed as personal copyright (see §§ 12 ff. UrhG). It consists of a personal and property right component. Both components are inseparably connected according to German understanding. In contrast, the "copyright" gives the author rights to reproduce or distribute, the mainly exploitation-related rights, which are protected under German copyright law, and all in §§ 15 ff. UrhG are regulated.

This has the consequence that the German copyright law, as such, in contrast to the "Copyright", cannot be transferred in principle (§ 29 para. 1 UrhG). Instead, simple or exclusive rights of use are granted, which can go so far that the author himself is excluded from using his work. These rights of use can be also be granted as transferable rights under German law. Therefore only rights of use of the work, but not the copyright as such. Such rights of use are generally granted or transferred by means of a license agreement.

The "copyright", therefore, does not fully correspond to the German "Urheberrecht". If in OSS licenses, the majority are based on US-American license rights, and thus the "copyright" idea from the "transfer of the Copyright", is defined under German law as the granting transfer of rights of use. [9][1]

2.4.6 Copyleft

"Copyleft is a general method for making a program free software and requiring all modified and extended versions of the program to be free software as well. The simplest way to make a program free is to put it in the public domain, uncopyrighted. This allows people to share the program and their improvements, if they are so minded. But it also allows uncooperative people to convert the program into proprietary software. They can make changes, many or few, and distribute the result as a proprietary product. People who receive the program in that modified form do not have the freedom that the original author gave them; the middleman has stripped it away."[10]

2.4.7 Contributor

A contributor represents "each individual or entity that creates or contributes to the creation of Modifications"[11].[12]

2.4.8 Contribution

The contribution represents either the "initial code and documentation under this Agreement" or the program's changes and additions. "Contributions do not include additions to the

[1] Section "6.4.1 Abgrenzung zum anglo-amerikanischen Copyright"[9] translated by the translation tool DeepL.

Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program." [13][14]

2.4.9 Propagation

"To 'propagate' a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well." [15] [16]

2.4.10 Convey

"To 'convey' a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying." [15] [16]

2.4.11 Covered Software

Covered Software "means Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof." [12]

2.4.12 Covered work

Covered work represents original software, modified software, or a combination of both. [11][15][16]

2.4.13 Combined Work

"A 'Combined Work' is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the 'Linked Version'." [17]

2.4.14 Source Form

The source form is a "common form of computer software code in which modifications are made"[11]. [12][15][16]

2.4.15 Executable Form

The executable form represents "any form other than source"[12][11]. Further commonly used terms are "object code"[15][16] or "non-source form"[15][16].

2.4.16 Larger Work

Larger work represents work that combines covered software or parts thereof with other work not governed by the terms of this software. [12][11]

2.4.17 Unmodified / Original / Pristine

"Original Software means the Source Code and Executable form of computer software code that is originally released under this License." [11]

2.4.18 Modifications

To "modify" a work means to add, delete or modify something of the original. It can also be a new file that contains parts of the original software.[12][11] "The resulting work is called a 'modified version' of the earlier work or a work 'based on' the earlier work." [15][16] An example for modifying is fixing a bugfix.

2.4.19 Derivative Work

A derivative work represents a substantial, more significant change in comparison to the modification. An example would be the release of a new version with new features.

2.4.20 Program

A program defines the distributed work licensed under this license. [11][13][15][16]

2.4.21 Disclaimer

License agreements may contain disclaimers regarding liability and warranty. These texts are usually consistent.

2.4.22 Patent Claims

Patent claims "means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor."[11]

3 Classification and Objectives

3.1 Context

This thesis can be assigned to the subject field of business information systems. When planning the Walking Skeleton of a software product, the main focus is defining and then integrating the Technology Stack. It extends the Technology Platform, which consists of a programming language, an optional run-time environment, and a standard library, with frameworks and libraries[3]. The platform has to be selected, specified once, and never changes. The four most popular platforms in practice are Java, JavaScript, .NET, and SAP. From .NET and SAP, the problem concerns the provider and not the user, so using the associated libraries and frameworks does not usually cause problems. On the other hand, considering Java and JavaScript, there are a large number of frameworks and libraries that may be used in combination. The responsibility for compliance with the license terms and conditions is with the users themselves.

A characteristic of software is the unlimited nesting and reuse of components. Nesting has increased in recent years due to the massive reusing of OSS components. To develop a software product without the use of OSS software is almost impossible nowadays. We are in a new software era, in which it is no longer possible to achieve the required time to market without the use of OSS components and, therefore, can no longer be competitive. The strong nesting of OSS components by the many transitive dependencies increase the complexity significantly.

Each of these components has one or several licenses, specifying the terms of use of the respective component. Thus, the thesis can also be assigned to the field of licensing law.

In small companies (50-100 employees, single location), these challenges can still be tackled by personal communication. However, as the number of employees and distributed locations increases, the tackling of these challenges becomes more and more complicated. Therefore, a central enterprise information base is advised for supporting the resolution of the mentioned challenges in practice.

3.2 Requirements

3.2.1 General Requirements

Architects need help with the selection of OSS components. For this purpose, they need a meaningful basis for decision making that indicates whether the use of a particular

component is legally permitted. Since software components are highly nested and contain many transitive dependencies, the goal is to create a common database. Therefore the manual effort is reduced, as components only have to be configured once and can be reused several times.

Additionally, the database contains licenses, which also have to be configured once and can be reused repeatedly. Since the data is sensitive and correctness must be ensured, the users are assigned to different roles to prevent the data's manipulations. Besides, the evaluation should also consider the usage of the components, which can influence the license terms that must be fulfilled. For example, internal use is often harmless and does not require any conditions, while a distribution implies many restrictions. This bachelor thesis only provides a basis for the design and does not cover the integration into the application build process.

Furthermore, there should be the possibility to request legal assistance. Certain cases require additional manual evaluation, which requires cooperation between architects and legal experts. Even legal experts often have difficulties reading the license texts since they are usually based on American law and written by architects. Furthermore, they contain specific information about the architectural design of components, which an architect understands better. Additionally, the tool should provide legal experts with a structured overview of the licenses to use this data as a basis for the evaluation.

3.2.2 User Requirements from Interviews

In order to identify the users' pain points and requirements, nine interviews were conducted with four architects, an OSS expert (OE), a product owner (PO), a product manager, and two legal experts (LE) from the legal department of msg [1].

Questions

The following questions were prepared before the survey. However, the survey also considered the respondents' interests, and based on their answers, further questions were asked. Some of the interviewees gave an insight into the current processes.

1. What is your role in projects?
2. How educated are you on OSS?
3. How high is the percentage of open source components in the project?
4. Who decides which OSS components are used? How do you choose them?
5. When do you deal with Open Source Compliance?
6. What problems have you experienced regarding OSS?

[1] msg is an independent, internationally active group of autonomous companies with more than 8,000 employees. The core competency: intelligent IT and industry solutions.[18]

7. Did you use the current solution from msg (msg OSS analysis database)? What are the advantages and disadvantages?
8. What would you expect to find in the OSS portal? Why?
9. Do you understand the UseType Descriptions?
10. Do you understand the Obligation Descriptions?
11. Is the display of the result as a five-color traffic light system useful? How can it be improved?
12. Do you have any further questions or comments?

Results

The users are grouped into personas to keep the answers anonymous. Requirements of the architects, who mainly develop standard software solutions, were different from those who establish individual software, which is why they are grouped into two roles: Architect-Product (AP) and Architect-Individual (AI).

Requirement	AP	AI	OE	LE	PO
Overview of used Components	✓	✗	✗	✗	✗
Summary of license information	✓	✓	✗	✓	✓
Notifications for security issues	✓	✓	✗	✗	✓
Notifications for component updates	✓	✓	✗	✗	✓
Automatic generation of reports	✓	✓	✗	✗	✓
Automatic evaluation of transitive dependencies	✓	✓	✓	✗	✗
Build-process integration	✓	✓	✗	✗	✓

Table 3.1: Overview of requirements

- **Architect of individual software solutions (AI)**

  An architect who mainly develops individual software solutions does not usually deal with the verification of licenses because the customer usually provides a white list, a list of allowed OSS components. He focuses on the quality of the components and is not concerned with license texts, and he does not even know if he is responsible for checking the licenses since there is no defined process. Furthermore, his impression is that it is time-consuming, and he finds it difficult to check the licenses as he is not familiar with that particular discipline. His expectations from a tool would be that the process is fast and easy. A summary of a license that says what is allowed and what is not would help him a lot. Ideally, the process should fit into the usual development environment, i.e., the build process. Furthermore, notifications that warn about security issues and components updates would be helpful.

- **Architect of standard software solutions (AP)**

  An architect who mainly develops standard software solutions considers himself responsible for selecting licenses and the license compliance check, even if the license compliance process is not his passion. In addition to good quality components, it is also essential to deliver software to the clients that comply with the license terms. He is familiar with licenses and assists others with this task if requested. However, he does not check all transitive dependencies, as this is very time-consuming. Furthermore, he desires an easy and fast process, and prefers the integration into the build pipeline and importing the Maven pom.xml to automate the process.

- **OSS Expert (OE)**

  His work includes answering questions about OSS and collecting information about the components of OSS with their licenses. Therefore an automatically generated report which contains all used components with their respective licenses would be beneficial. Furthermore, he desires a tool that supports the process and the support, since others' support takes much time.

- **Legal Expert (LE)**

  A legal expert deals with OSS issues in addition to several other legal aspects and provides support in this process on request. Understanding license texts are also challenging for legal experts, as they are based on American law and written by architects. The desire is an attractive tool that includes a license checker, which automatically extracts all rights, obligations, and prohibitions from the license texts. Furthermore, the tool should contain a note that clearly states that this check does not replace legal advice.

- **Product Owner (PO)**

  The Product Owner is interested in delivering a high-quality product to the customer, so ensuring that the architects comply with the license regulations is essential. They would require the tool to have a product overview with all components and licenses and to receive notifications on security issues and component updates.

- **Product Manager**

  The Product Manager has no interest in using such a tool and therefore has no requirements.

3.3 Related Work

3.3.1 Black Duck Suite

"Black Duck provides a comprehensive software composition analysis (SCA) solution for managing security, quality, and license compliance risk that comes from the use of open source and third-party code in applications and containers." [19] Firstly, it identifies OSS components by build process monitoring, file system scanning, and code snippet matching. Secondly, it detects security vulnerabilities and notifies users directly. Furthermore, the tool provides DevOps Integrations to define agile development teams' processes and ensure that applications comply with open source license terms and that the components are free of vulnerabilities. [19]

3.3.2 WhiteSource

WhiteSource is a tool that automatically identifies components and their transitive dependencies according to license compliance and security vulnerabilities. The tool is integrated into the build process. Furthermore, automatic policies can be defined to control usage. Therefore, a blacklist can be assigned to reject licenses automatically, a white list to automatically approve licenses and a list of licenses that need to be approved. When a checked component or dependency is licensed under a blacklisted license, the user gets an alert and/or the build will fail. [20]

3.3.3 Fossa

Fossa is an Open Source Management Tool that scans the full code to check whether license obligations are fulfilled, and vulnerabilities are detected. If it is integrated into the build process, the check will be done at every commit. For product evaluation, specific evaluation policies are defined to categorize license into the following categories: Uncategorized, Deny, Flag for Review, Approve. [21]

3.3.4 spdx

"The SPDX License List is a list of commonly found licenses and exceptions used in free and open source and other collaborative software or documentation. The purpose of the SPDX License List is to enable easy and efficient identification of such licenses and exceptions in an SPDX document, in source files or elsewhere. The SPDX License List includes a standardized short identifier, full name, vetted license text including matching guidelines markup as appropriate, and a canonical permanent URL for each license and exception." [22]

3.3.5 tl;dr Legal

Tl;dt Legal is a website where Software Licenses are explained in plain English. An explanation contains a short description and a summary that contains what is allowed to do, what is

not allowed, and which obligations must be fulfilled. [23]

3.3.6 OpenLogic Stack Builder

OpenLogic is a tool that assembles a technology stack based on short questions in order to meet the requirements. For example, to select a platform, the user is asked whether he is more interested in VM or container-based solutions. Furthermore, the user has to specify whether his solution is a large, multi-regional deployment or a single data center or availability zone. [24]

3.3.7 Nexus Vulnerability Scanner

Nexus Vulnerability Scanner is a tool to scan the source code of a project in order to discover vulnerabilities or license risks for components and transitive dependencies. "Licenses are categorized as Copyleft (red), Non-standard or Not Provided (orange), Weak Copyleft (yellow), and Liberal (blue)". The scanner checks for license declarations in the source and determines the risk level. The result of the evaluation contains a value between one and ten and is grouped into four categories for visualization as a doughnut chart: No Threat (0), Moderate (1-3), Severe (4-7), Critical (8-10). Vulnerabilities correspond to the Common Vulnerability Scoring System (CVSS) score. Furthermore, security and license risks of the current version are compared with newer versions. After the evaluation, a report is provided with the information.

3.3.8 Eclipse SW360

"SW360 is an open source software project licensed under the EPL-1.0 that provides both a Web application and a repository to collect, organize and make available information about software components. It establishes a central hub for software components in an organization. SW360 allows for: tracking components used by a project/product, assessing security vulnerabilities, maintaining license obligations, enforcing policies, and generating legal documents, Integration with other tools and data sources (e.g. license scanner, static code analysis, build infrastructure, etc" [25]. A build fails if a license obligation is not fulfilled, e.g., when the license text must be provided, but cannot be found. Furthermore, reports are provided with information about used components and their licenses. [25]

3.3.9 FOSSology

"FOSSology is a framework, toolbox and Web server application for examining software packages in a multi-user environment. A user can upload individual files or entire software packages. FOSSology will unpack this upload if necessary and run a chosen set of agents on every file of the upload. An agent can implement any analysis operation on a text file. The FOSSology package as of now focuses on license relevant data. However, it could be extended with analyses for different purposes (e.g. static code analysis)." [26]

3.3.10 LChecker

LChecker is a tool developed for automatic checking of license compliance. The tool "utilizes Google Code Search service to check whether a local file exists in an OSS project and whether the licenses are compatible." [27]

3.3.11 Kenen

Kenen is a process to support OSS license compliance for java projects. It contains a repository with pre-approved components. An analysis tool called Joa finds matches of the source (.java) and binary (.class) files. If Joa detects a match, the component is approved. In order to identify a license of a component, a pattern-matching tool called Ninka is used. [28]

3.3.12 Method of License Compliance of OSS Governance

Licenses of software are detected. Therefore, the name of the license, keywords, or the complete license text is being searched for. An algorithm checks whether the licenses are compatible or not. Only the compatibility is evaluated, but there is no evaluation of the usage. [29]

3.3.13 Suggestion for an easy to use tool

A matrix is suggested to help to meet the requirements and improve the communication between developers/software architects and legal professionals. The obligations depend on the usage of the component.

"According GPL v2 Section 2, one is allowed to modify copies of the program and to copy and distribute such modifications or work, provided, a) that the modified files carry prominent notices stating who changed the files and the date of any change, and – the most famous requirement in Open Source Licensing – b) that any work that one distributes or publishes, that in whole or in part contains or is derived from the program or any part thereof to be licensed as a whole at no charge to all third parties under the terms of the GNU General Public License, Version 2 (GPL-2.0-only) License. This means, if some GPL-2.0-only-licensed source is taken and built into a proprietary source, then, when distributing such combined work, the requirement may arise, that also the combined proprietary source has to be put under GPL-2.0-only. This might be a bit touch-and-go, as, perhaps, it is not intended to disclose a company's intellectual property. However, not every software component distributed together with GPL-2.0-only-licensed software falls under these requirements: In the case, the further software component is an independent and separate work, it may remain distributed under its own license. According [4] a work is regarded independent when it communicates with the GPL-2.0-only-licensed program, e.g., via an operating system interface, as, e.g., pipe, socket or command line. Taking these few points together, it can be seen that, when using GPL v2-licensed software, the legal consequences depend strongly on what is done technically." [30]

Therefore, a matrix is used where the developers provide component information, and they can x-check several modes to specify the usage. The legal professionals evaluate that information, and the result contains a traffic light and x-checks in the columns with the obligations which must be fulfilled. [30]

3.3.14 Open Source Analysis Database (msg)

The Open Source Analysis Database was developed in 2012 by msg [2] in cooperation with an external lawyer, who is specialized in OSS License Compliance. The method offers an Excel-based solution for checking the license conditions.

The architects can enter any amount of specific product information, and they receive a concrete evaluation for each component. In addition to the component, it is necessary to specify how the component was used. For the usage 14 use types (Table 4.26) have been defined, which are mentioned in the license texts. Besides the use types, 13 obligations (Table 4.27) were defined, which model the license's terms. If an obligation must be fulfilled, often depends on the usage, which is why an adjacency matrix represents the license terms. For example, some OSS licenses state that the use type "Modification of the source code" is only allowed under the condition that "The modifications are made available to everyone again under the same OSS license", then the cell from the intersection of the particular use type and the particular obligation contains an entry.

After specifying how the component was used, an algorithm calculates a value that is based on the Defense Readiness Condition (DEFCON) of the American military: from "DEFCON 5" for harmless conditions to "DEFCON 1" for KO conditions. This method provides a good basis by mapping the licenses and components. However, there is potential for improvement since the method does not include transitive dependencies, and there is no central tool. The Excel file is located in one division, and therefore the database cannot be shared within the whole company, which would reduce the manual effort. Therefore, this bachelor thesis's solution deals with an elaboration based on this method considering transitive dependencies and a central database. [1]

[2] msg is an independent, internationally active group of autonomous companies with more than 8,000 employees. The core competency: intelligent IT and industry solutions.[18]

4 Concept

4.1 Introduction

The brain icon indicates own ideas or improvements, as the method is based on an already existing method.

4.2 Data model

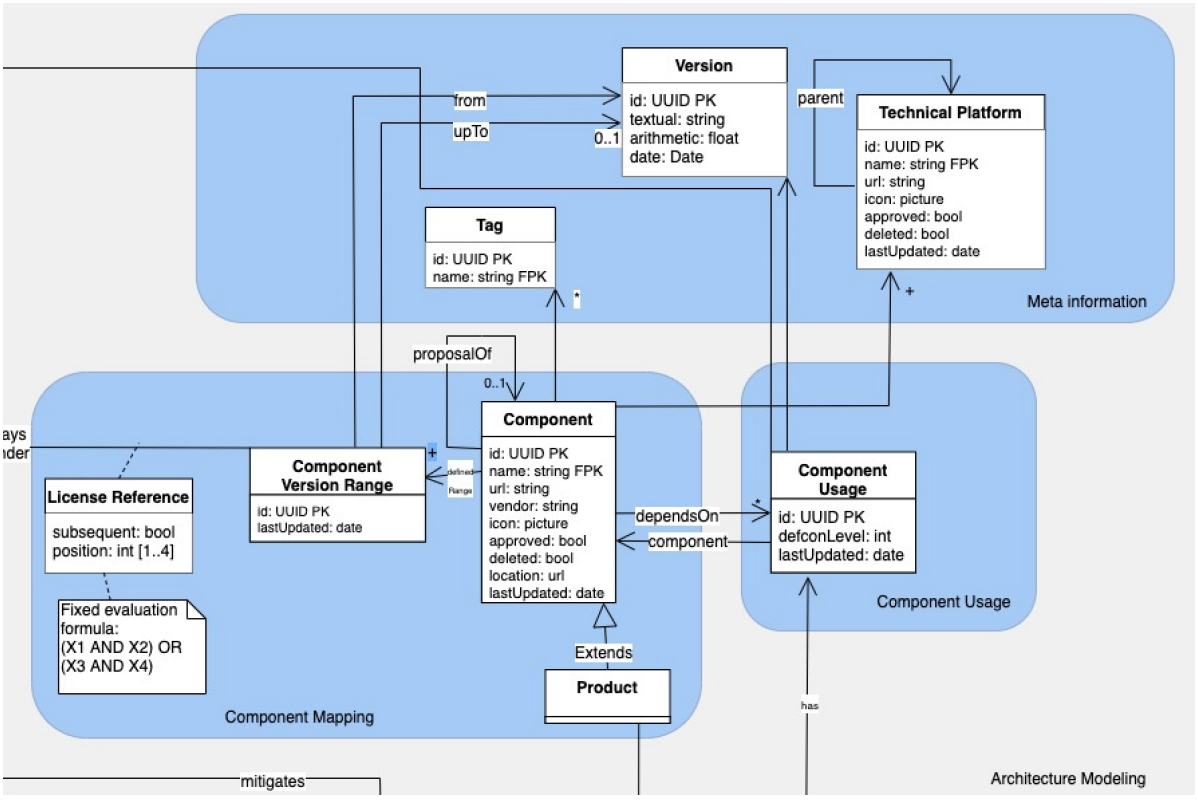


Figure 4.1: Architecture Modeling

Note: Normal notation represents attributes, and Italic represents relationships.

- **ComponentUsage**: A component usage specifies the component which is used with the particular version. Addtionally, it contains information on the actual usage of the component.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain-specific primary key
defconLevel	The defconLevel represents the result of the evaluation as an integer. The range goes from one to seven and indicates how bad the conditions are. Thereby the value one represents a knockout criterion and the value seven is harmless.	The evaluated value is stored in the database to provide the information to the user. The name defcon was derived for historical reasons.
lastUpdated	Date when the defconLevel was last evaluated.	The user is able to recalculate the status after pressing a button. When information change in the system affects the evaluated value of the component usage, the user should get notice and get a new evaluation. When the evaluation would be updated without the user recongnizing, it could lead to surprises.
component	OSS Component, which is used for the development of the product.	The architect wants to use the component and needs to know whether it is allowed or not.
version	Specifies the version in which the component is used.	The version is relevant to determine the appropriate version range of the component in order to determine the license information which is relevant for the evaluation.
specifies	Specifies how the component is used in the product.	The usage is relevant for the evaluation of the component usage.

- **Component**: A component represents a software component as defined in the Foundation. This is an OSS component that can be used in the development of a product and for which the architect wants to know whether he can use it.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
name	Designation	For recognition and basis for discussion
url	Link to component	To find the original component
vendor	Author of component	Part of component information
icon	Icon of the component	Recognizing
approved	Indicates whether the component has been approved. In case information have been changed, the component information must be improved in order to keep correct information.	After the component has been approved the changes are visible to any user of the system and the old version is replaced by the changes.
lastUpdated	Date when the component was last updated	If it has not been edited for a particular time and the element is not being referenced, it can be deleted to prevent data garbage
dependsOn	Transitive dependencies of a component	A component often has multiple transitive dependencies which must be considered in the evaluation.
definedRange	Range for which the license modeling applies	A new version of a component often contains small changes that do not affect license modeling. Therefore, the license modelling is defined for a specific license range in order to reduce effort to conduct the information and to avoid duplicates.

technicalPlatform	Platform of component	Can be used to recommend only components which are compatible with the desired platform
tag	Provides information about the component	Can be used for recommendations
proposalOf	Proposed change for a component	When an item is modified, a copy is created and stored until the change is approved

- **Version**: Represents a version of a component or product.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
textual	Textual representation of a version	A version often has the format x.y.z
arithmetic	Arithmetic number of a version	To be able to compare a version with another
date	Publish date of version of component/license/product	To be to check whether it is the newest version

- **ComponentVersionRange**: A version range of a component is defined so that the license modelling does not have to be redefined for each new minor version. For example, if a new version only contains bug fixes or more significant features are added without using another component, and the license of the component does not change, the version range is extended.

Subject	**Purpose (What?)**	**Reason (Why?)**
componentRange	Component is defined for a range of versions	No need to create a new component for small changes that are not essential for the calcualation of the DEFCON
id	Technical unique identifier	Not all entities have domain specific primary key
from	First version for which the license modeling is defined	To define a range and check whether a component is within this range

upTp	Last version for which the license modeling is defined	To define a range and check whether a component is within this range
staysUnder	A component can be under several licenses	Dual licensing is a frequently used case which must be considered

- **Product**: A product represents a software product for which an architect wants to verify compliance with the license terms.

Subject	**Purpose (What?)**	**Reason (Why?)**
extends	A product inherits from a component	It has the same properties as a component
document	A product can have multiple attached documents	To attach important documents of a product

- **TechnicalPlatform**: The Technical Platform depicts a specific property of a component or product. This information can be used, for example, to give users recommendations about the most commonly used components. However, if the user is working in a Java environment, he or she does not want to have JavaScript recommendations, so this information is relevant.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
name	Designation	For recognition and basis for discussion
url	Link to technical platform	To find the original platform
icon	Icon of the platform	Recognizing
approved	Shows whether the technical platform has been approved	After approval the change is visible to anyone
lastUpdated	Date when the platform was last updated	If it has not been edited for a particular time and the element is not being referenced, it can be deleted to prevent data garbage

parent	Parent of the technical platform (e.g. Java (Java EE, Spring), JS (Angular, nodejs, react, vue))	To further limit the range of recommendations

- **Tag**: A tag represents information and clusters components.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
name	Designation	For recognition and basis for discussion

- **LicenseReference**: A license reference is created when a new license is added to the component. It provides information about whether subsequent versions of the license are also valid and specifies the position relevant to the associated formula that was defined for evaluating the licenses.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
subsequent	Indicates whether subsequent versions apply	It needs to be considered in the evaluation
position	Position of the fixed formula to evaluate the legal status of the componentUsage: (X1 AND X2) OR (X3 AND X4)	The position is relevant to handle dual licensing and distinguish between "and" and "or" operations

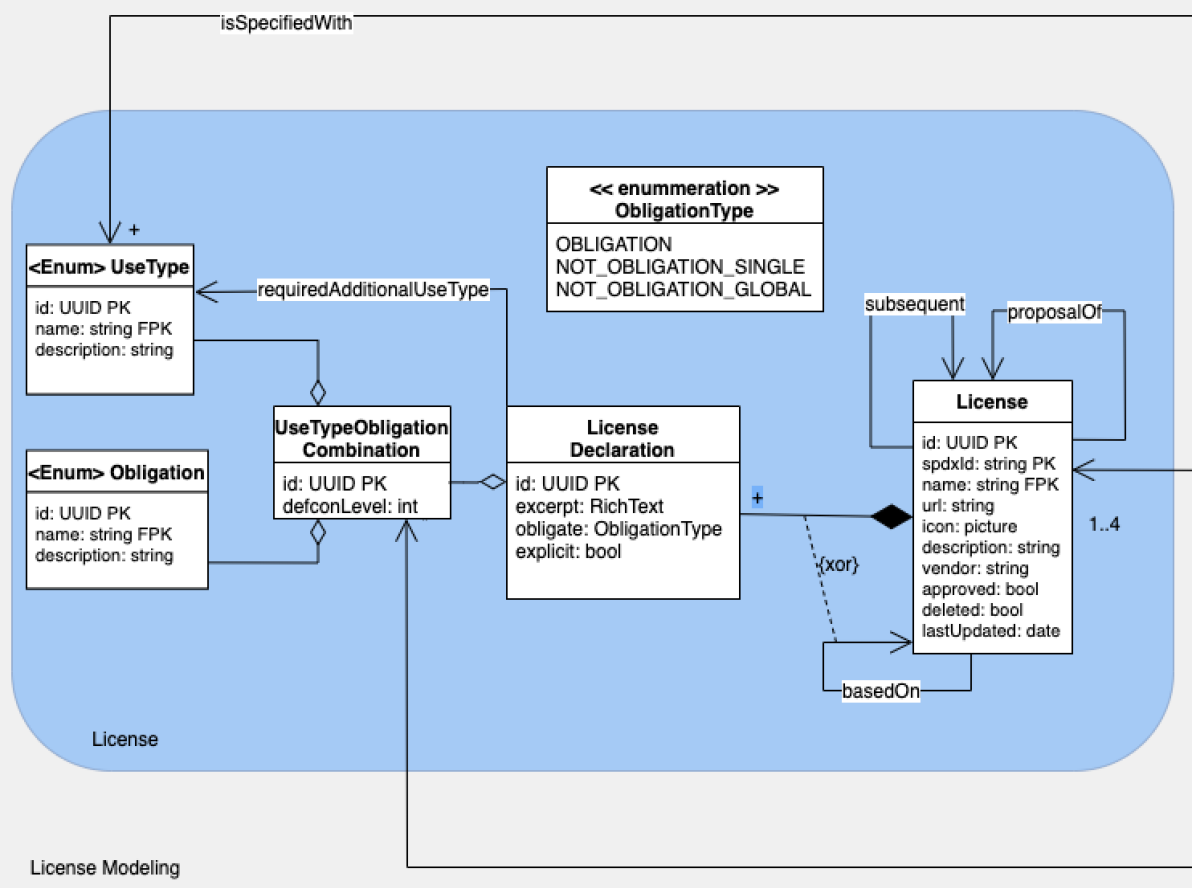


Figure 4.2: License Modeling

- **License**: A license is a written document that defines the terms of use for the respective components under this license. The information of a license, as well as its contents, is relevant for the evaluation of a specific component.

Subject	Purpose (What?)	Reason (Why?)
id	Technical unique identifier	Not all entities have domain specific primary key
spdx	General identifier for licenses	Common used license identifier
name	Designation	For recognition and basis for discussion
url	Link to license text	To find the particular license text
icon	License Icon	Recognizing
description	Prose summary of a license text	To summarise the license terms

vendor	Author of the license	Part of license information
approved	Shows whether the license has been approved	After approval it is visible to anyone
deleted	In case the license has been deleted there will be a "deleted" flag for the license to let the users know.	The license cannot be deleted if there are elements with references to this license.
lastUpdated	Date when the license was last updated	If it has not been edited for a particular time and the element is not being referenced, it can be deleted to prevent data garbage
basedOn	A license can be based on another license. For example, some license texts are copied and only the name is changed. Examples are shown in table Table 4.10.	Licenses with equal license declarations can be grouped in order to refer to the same license declarations.
subsequent	Subsequent versions of a license	Sometimes you can choose from the specific license and all subsequent licenses
proposalOf	When license information have been changed a copy is created which must be approved.	Changes must be improved in order to replace the previous information.
consistsOf	A license consists of multiple license declarations.	Major points of legal texts can be reduced to a set of License Declarations.

License	**Derived Licenses**
MIT	The JSON License[31], Prototype License[32], Scriptaculous[33], Licensing terms for SLF4J[34], W3C SOFTWARE NOTICE AND LICENSE[35]
BSDv3C	ASM[36], HSQLDB[37], JAMon[38], postgresql[39]
BSDv4C	dom4j[40], FreeMarker[41]
MPL	gSoap[42]
SDN	MAXDB[43]

Table 4.10: Derived Licenses

- **LicenseDeclaration**: A license declaration represents a statement of a license according to a specific obligation that requires a particular usage. It can either contain a requirement that must be met or excludes a requirement due to the particular usage.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
excerpt	Indicates the excerpt of the license text from which the information of the LicenseDeclaration is derived.	The license excerpt is a relevant information for the license modelling.
obligate	Indicates whether an obligation must be fulfilled, excluded or the whole license terms do not need to complied with.	The information is relevant for the evaluation in order to decide whether an obligation must be fulfilled or not.
explicit	Indicates whether the information is explicitly or derived from another text section	For the complete collection of the data with the corresponding source
belongsToUseType-ObligationCombination	Each license declaration refers to the corresponding value according to the specific use type and obligation as shown in Figure 4.26.	The object contains a value which is relevant for the evaluation of a componentUsage.
requiresAdditional UseType	Specifies an additional use type, which describes a usage which cannot be described by a single use type.	Some license terms exclude an obligation when the component is used in a specific way, and usage is sometimes covered by two different use types in the license modelling which is why an additional use type is required.

- **«enumeration» ObligationType**: The ObligationType indicates whether it is an obligation, where the user needs to fulfill a burden, or if a specific use type excludes a single obligation or the whole usage.

Subject	Purpose (What?)	Reason (Why?)
OBLIGATION	Represents the state that a condition must be fulfilled in order to comply with the license terms.	A license agreement contains several conditions which need to be fulfilled.
NOT_ OBLIGATION_ SINGLE	Represents the state that a condition is excluded due to the usage.	When a specific usage is not restricted, the obligation must not be fulfilled in order to comply with the license terms.
NOT_ OBLIGATION_ GLOBAL	Represents the state that all license terms can be excluded due to a specific use type.	Some license agreements include the statement that the license terms do not apply when the component is used in a specified way.

- **UseTypeObligationCombination**: Specifies a relevant value for the evaluation that applies to a particular obligation for a specific usage.

Subject	Purpose (What?)	Reason (Why?)
id	Technical unique identifier	Not all entities have domain specific primary key
defconLevel	Defcon level of UseTypeConditionCombination	Use types are of a different severity and must be considered individually
belongsToUseType	Specifies the use type of the corresponding license declaration for which the value of the UseTypeObligationCombination applies.	Specifies for which purpose the conditions apply.

belongsToObligation	Specifies the obligation of the corresponding UseTypeObligationCombination for which the value applies.	Specifies the corresponding obligation for the license declaration.

- **UseType**: List of usage types under which the licenses distinguish and on which they make a statement.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
name	Designation	For recognition and basis for discussion
description	Description of a use type	Explanation for the user, so that they can correctly classify the use.

- **Obligation**: List of obligations under which the licenses distinguish and on which they make a statement.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
name	Designation	For recognition and basis for discussion
description	Description of a condition	Explanation for the user, so that they understand what they need to do in order to comply with the license terms.

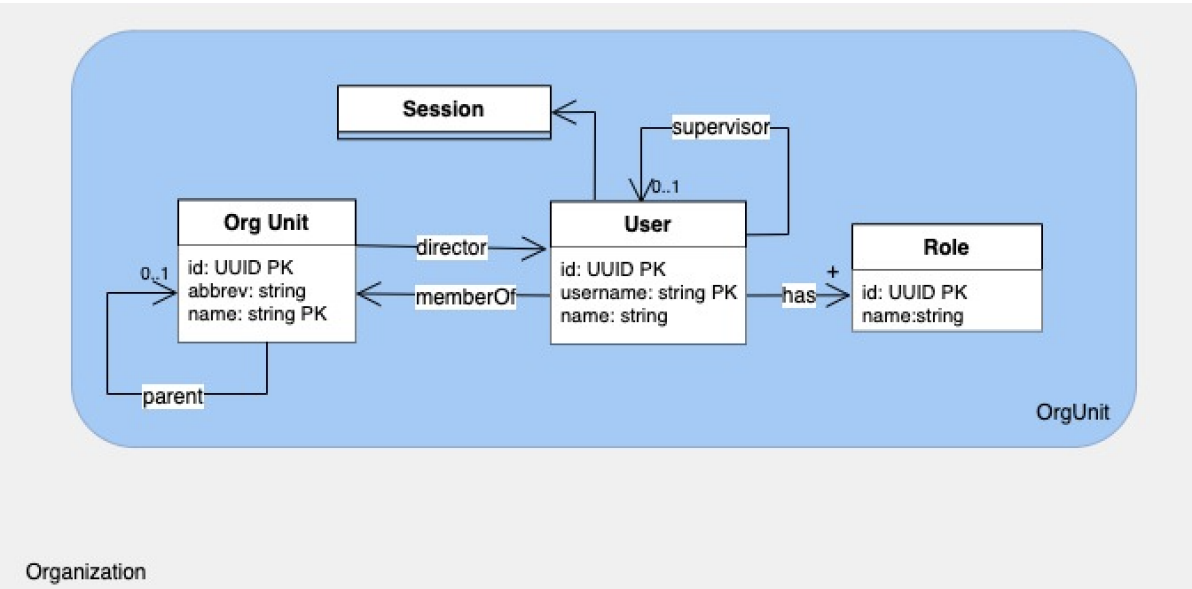


Figure 4.3: Compliance Process

- **Mitigation**: Once the calculation has resulted in a list of obligations that must be fulfilled, the user must be able to heal them by fulfilling them. Mitigation is created once the user has fulfilled an obligation.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
partially	Shows whether the burden has been partially fulfilled	Sometimes the burdens cannot be fully fulfilled
comment	Comment for the user	For example, to keep a information to remember something (e.g. comments for a decision)
mitigates	A mitigation is specified with a license declaration where the corresponding obligation has been fulfilled.	The mitigation is relevant for the evaluation for which the defconLevel of the useType-ObligationCombination is a basis.

has	The mitigation is related to the componentUsage for which the obligation has been fulfilled.	An obligation can only be fulfilled for a single component usage.

- **Document**: A document can be attached to a product. It can be a report, which is automatically generated by the system, or an evaluation, which is attached to save all information to one location.

Subject	**Purpose (What?)**	**Reason (Why?)**
data	Date when the document was created	Keep the history and find the newest version.
read	Shows whether a document has been read	To highlight unread notifications
title	Title of a document	For recognition and basis for discussion
data	Data of the document (e.g. pdf)	A document consists of data, keep history of a product, provide documents to customers

- **Review**: Represents a review of the product, which is optional if one or more components show problems according to the calculation of the system and the architects need further legal advice.

Subject	**Purpose (What?)**	**Reason (Why?)**
extends	A review inherits from a component	It has the same properties as a document

- **Human**: Represents a review, which is accomplished by an expert.

Subject	**Purpose (What?)**	**Reason (Why?)**
extends	A review human review inherits from a review	It has the same properties as a review

- **General**: Represents a review, which is automatically generated by the system.

Subject	Purpose (What?)	Reason (Why?)
extends	A general review inherits from a review	It has the same properties as a review

- **Report**: Represents a report that is automatically generated by the system. This report can contain information about the components used, as well as the associated licenses and their transitive dependencies.

Subject	Purpose (What?)	Reason (Why?)
extends	A report inherits from a component	It has the same properties as a document

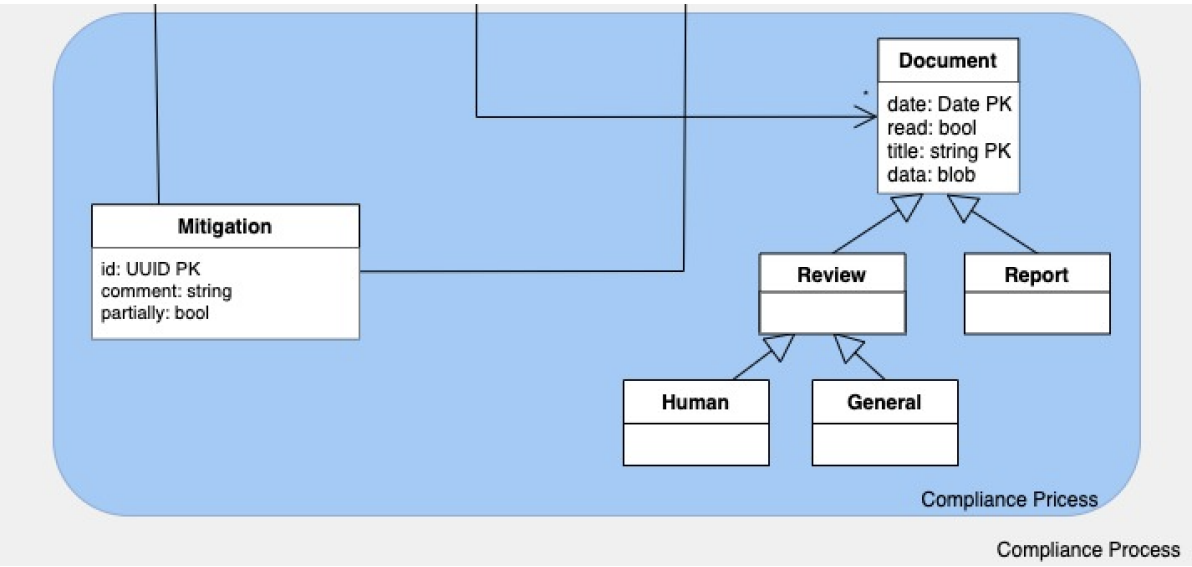


Figure 4.4: Organization

- **OrgUnit**:Represents any kind of organization unit, can also be a project.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
abbrev	Abbreviation of OrgUnit	Part of OrgUnit information
name	Designation	For recognition and basis for discussion
parent	Parent of OrgUnit	To represent the hierarchy of the organization.
director	Director of OrgUnit	A director holds the rights for his unit

- **User**: Represents any user of the system.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key

username	Unique assigned username	Required to login, cannot change
name	Represents the real name of the user	User can edit this name (e.g. after a marriage or do add an title)
supervisor	Supervisor of user	Person who manages the rights of the user
memberOf	OrgUnit to which the user belongs	Within an OrgUnit the user has special rights
has	Role/s of a user	Relevant for the authorization in the system
session	User session	Technically required to handle multiple HTTP requests as a single session

- **Role**: A role represents the user's role required for the logic of the server-side and the appropriate display of the user interface.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key
name	Designation	For recognition and basis for discussion

- **Session**: Represents a user session which is technical required to handle multiple http requests as a single session.

Subject	**Purpose (What?)**	**Reason (Why?)**
id	Technical unique identifier	Not all entities have domain specific primary key

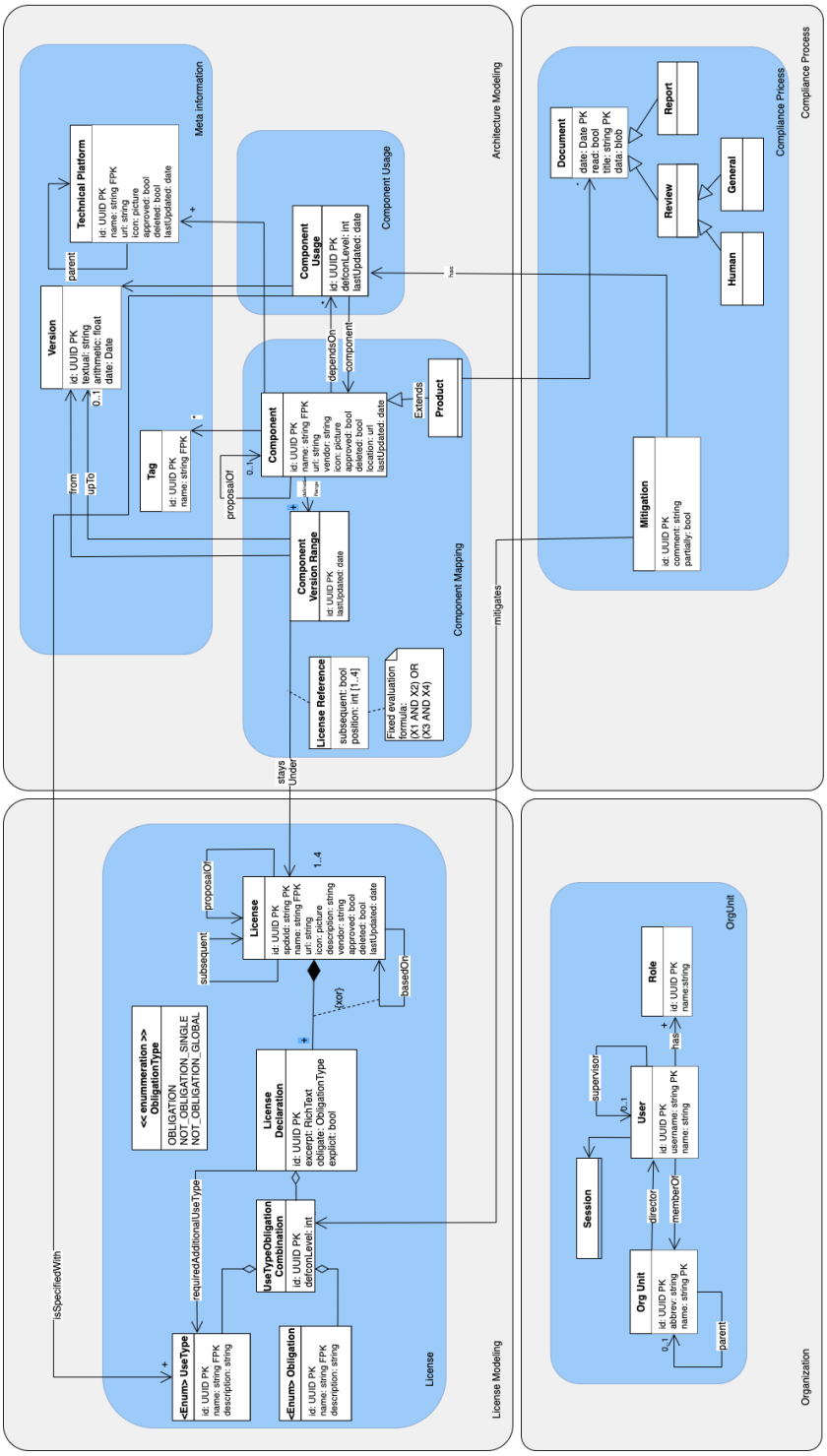


Figure 4.5: Data model

4.3 License Modelling

4.3.1 General

The modeling is primarily based on adjacency matrices of use types and obligations. Some OSS licenses, for example, contain the statement that the type of use "modification of the source code" is only allowed if the condition "the modifications are made available to everyone again under the same OSS license" is fulfilled. In such a case, the adjacency matrix contains an entry at the intersection of the specific use type and corresponding obligation. The licenses is defined by 14 use types (Table 6.25: Use Types) and 13 use types (Table 6.26: Obligations)[1].

Use Type	Description	Example
format: source	component is provided in pristine source format	WEB-INF/jquery.js
format: compiled	component is provided in compiled/converted/compressed format	com/example/foo.class
dependency: optional	component is loaded on demand and product would reasonably work without it	JDBC driver
dependency: mandatory	component is loaded/linked dynamically/statically and product does not function without it	Hibernate ORM
delivery: internal	component is used internally without distribution to other legal entities (e.g. built-time components)	Gradle/Ant/Maven
delivery: distributed	component is distributed to other legal entities (e.g. runtime components)	lib/example-1.2.3.jar
usage: local-call	component (via product) is locally called for execution	C:*Applications**Example**example.jar*
usage: remote-call	component (via product) is remotely called for execution (SaaS)	service.example.com:1234
communication: process	component is called from product via direct in-process mechanism (function call, dispatch table, etc)	component_function()

communication: system	component is called from product via system/network service (pipe, socket, dlsym, execve, etc)	/var/run/ component.socket
bundling: standalone	component artifacts still provided fully standalone and recognizable as such	example-1.2.3.jar
bundling: embedded	component artifacts embedded into product and/or not obviously recognizable	product-1.2.3.ear/ example-1.2.3.jar
artifact: pristine	component artifacts are all as-is, i.e. exactly as originally received from upstream vendor	example-1.2.3.jar!com/ example/foo.class
artifact: modified	component artifacts were added/replaced/removed	example-1.2.3.jar!com /example/addon.class

Table 4.26: Use Types

Obligation ID	**Obligation**	**Description**
NO-LIABILITY	No Liability	You cannot make the author of the component liable for any damages it causes.
KEEP-COPYRIGHT	Keep Copyright Information	The copyright information of the component's author has to be kept.
PROVIDE-LICENSE	Provide License Text	You have to provide the full license text of the component.
PROVIDE-SOURCE	Provide Source Code	You have to provide the full source code of the component.
ADV-CLAUSE	Advertizement Clause	Documentation and/or application has to show hint to the component (and its author).

RENAME	Name Change Required	The name of the component has to be changed (in case of modifications + redistribution).
NO-RELICENSE	No Relicensing Allowed	The component cannot be relicensed under a different custom license.
CTX-NON-MIL	Non-Military Use Only	Component is not allowed to be used in military or nuclear-power contexts.
CTX-NON-COM	Non-Commercial Use Only	Component is not allowed to be used in commercial contexts.
COPYLEFT-STRONG	Weak Copyleft Effect	License applies a weak/restricted copyleft effect.
COPYLEFT-WEAK	Strong Copyleft Effect	License applies a strong/-full copyleft effect.
NON-OSS-DEF	Non OSS Definition Compliant	License contains conditions which are not compliant to Open Source Software definition.
OTHER	Other Obligations	License contains arbitrary other major conditions not modeled/covered by us (fallback).

Table 4.27: Obligations

Each License Declaration represents a single cell of the adjacency matrix. The license analysis (see file license-modelling .xsxl on CD) provides the adjacency matrices of the following licenses: MIT [44], The 2-Clause BSD License (BSD-2-Clause) [45], The 3-Clause BSD License (BSD-3-Clause) [46], The 4-Clause BSD License (BSD-4-Clause) [47], ICU License (ICU) [48], Apache Software License 1.1 (Apache-1.1) [49], Apache Software License 2.0 (Apache-2.0) [50], MOZILLA PUBLIC LICENSE VERSION 2.0 (MPL-2.0), COMMON DEVELOPMENT AND DISTRIBUTION LICENSE Version 1.0 (CDDL-1.0) [11], Common Public License, version 1.0 (CPL-1.0) [14], Eclipse Public License - v 1.0 (EPL-1.0) [13], GNU Lesser General Public License, Version 2 (LGPL-2.0-only) [51], GNU Lesser General Public License, Version 3 (LGPL-3.0-only) [17], GPL-2.0-only [52], GNU General Public License, Version 3 (GPL-3.0-only) [15], GNU Affero General Public License, Version 3 (AGPL-3.0) [16], Oracle Technology Network License Agreement (OTN) [53] and SAP Developers Network MaxDB License Agreement Version 1 (SDN) [54] (Table 6.27: Full License Names).

The selection of the licenses is based on practical experience (msg) as well as on two openly viewable reports. The first report, "Open source security and risk analysis report from 2020 (OSSRA)" is a study of 1250 codebases in 17 different industries. It covers, besides other topics, the most commonly used licenses. The second is a technology review, which was created by lawyers. According to the Table 4.29, the license analysis includes all licenses included in the list of the most used licenses. Additional licenses are analyzed to verify that the modeling covers all use types and obligations.

License Name	**SPDX Identifier**
The MIT License [44]	MIT
The 2-Clause BSD License [45]	BSD-2-Clause
The 3-Clause BSD License [46]	BSD-3-Clause
The 4-Clause BSD License [47]	BSD-4-Clause
ICU License [48]	ICU
Apache Software License 1.1 [49]	Apache-1.1
Apache License, Version 2.0 [50]	Apache-2.0
MOZILLA PUBLIC LICENSE VERSION 2.0 [12]	MPL-2.0
COMMON DEVELOPMENT AND DISTRIBUTION LICENSE Version 1.0 [11]	CDDL-1.0
Common Public License, version 1.0 [14]	CPL-1.0
Eclipse Public License - v 1.0 [13]	EPL-1.0
GNU Lesser General Public License, Version 2 [51]	LGPL-2.0-only
GNU Lesser General Public License, Version 3 [17]	LGPL-3.0-only
GNU General Public License, Version 2 [52]	GPL-2.0-only
GNU General Public License, Version 3 [15]	GPL-3.0-only
GNU Affero General Public License, Version 3 [16]	AGPL-3.0
Oracle Technology Network License Agreement [53]	N/A
SAP DEVELOPERS NETWORK –MaxDB LICENSE AGREEMENT Version 1 [54]	N/A

Table 4.28: Full License Names

License	**msg**	**OSSRA Report**	**Technology Review**
MIT	✓	✓	✓
BSD-2-Clause	✓	✗	✓
BSD-3-Clause	✓	✓	✓
Apache-2.0	✓	✓	✓
MPL-2.0	✓	✓	✗
CDDL-1.0	✓	✓	✗
EPL-1.0	✓	✓	✗

LGPL-2.0-only	✓	✓	✗
LGPL-3.0-only	✓	✓	✗
GPL-2.0-only	✓	✓	✓
GPL-3.0-only	✓	✓	✓
Other	✓	✗	✗

Table 4.29: Most frequently used licenses

4.3.2 License Analysis

The matrices of the license analysis are not just plain adjacency matrices. Each cell, which is usually identified by true or false, contains further information. Empty fields indicate that the license makes no statement about the particular obligation and the corresponding use type. Filled fields contain a text which represents an excerpt from the corresponding license text. That text indicates that the particular obligation applies according to the particular use type.

Additionally, each filled cell is colored. The color indicates further information. There are three colors, each with two different contrasts. Light indicates that the information is implicitly given from the stated license excerpt, and dark indicates that the information is explicitly given. If the information is explicitly given, the text's keywords that indicate the use type are marked bold. The three colors are blue, orange and red. The color blue indicates that the component is an obligation (OBLIGATION), which means something needs to be fulfilled.

Orange indicates a NOT_OBLIGATION_SINGLE. This states that the user does not have to fulfill the obligation if he uses the component in a certain way. For example, some licenses require the user to comply with conditions if the component is distributed. Since there is no statement about obligations for internal use, the usage is harmless, and the conditions do not have to be met.

The last case NOT_OBLIGATION_GLOBAL excludes not only one single obligation, but all obligations of the license. Some licenses state that the license terms do not apply in case of a specific usage.

- Observing the license analysis of the The MIT License (MIT) (Figure 4.6) license, the license contains only one color. The reason for this is that this license does not differentiate between the use types. The disclaimer "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"[44] e.g. states that the author or copyright holder is not liable for any damages regarding the use of the software component. This disclaimer is a general statement and does not refer to a specific use type, so it can be implied that this applies regardless of the use. Therefore the data is stored as implicit information for all use types. Referring to the data model, the disclaimer mentioned is assigned to the excerpt of a License Declaration, the obligation type is "OBLIGATION" and the boolean "explicit" is false.

	format: source	format: compiled	dependency: optional	dependency: mandatory	delivery: internal	delivery: distributed	usage: local	usage: remote-call	communication: process	communication: system	bundling: standalone	bundling: embedded	artifact: pristine	artifact: modified
NO-LIABILITY	**"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."	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR	"IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
KEEP-COPYRIGHT	"The above in all copies or **copyright notice** and this permission notice shall be included substantial portions of the Software. "	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included	"The above in all copies or copyright notice and this permission notice shall be included
PROVIDE-LICENSE	"The above in all copies or copyright notice and this **permission notice** shall be included substantial portions of the Software. "	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and	"The above in all copies or copyright notice and
PROVIDE-SOURCE														
ADV-CLAUSE														
RENAME														
NO-RELICENSE														
CTX-NON-MIL														
CTX-NON-COM														
COPYLEFT-STRONG														
COPYLEFT-WEAK														
NON-OSS-DEF														
OTHER														

Figure 4.6: MIT License Analysis [44]

- However, looking at the BSD-2-Clause license, which contains the same obligations, the different types of use are also considered. The disclaimer about NO-LIABILITY is also valid for all use types. However, the section "Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer"[45] explicitly states that you must keep copyright information when distributing the source code. Therefore there is an entry at the intersection of the use type "format: source" and the obligation "keep-copyright". Referring to the data model, the corresponding license declaration has an obligation type "OBLIGATION", and explicit is true. The section "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."[45] gives the information for the same obligation as above and the use type "format: compiled". The term "redistribution", which is used in both sections, gives further information. That means that these obligations must only be fulfilled if the code is redistributed, which is covered by the use type "delivery: distributed". It further indicates that there are no restrictions for internal use. Therefore,

the use type "delivery: internal" is excluded. The adjacency matrix illustrates that with the color orange with a light contrast, and the license declaration's obligation type has the value "NOT_OBLIGATION_SINGLE" and explicit is false. For all other use types, there are no specifications.

The first question, which still has to be clarified, is when a component is used internally. The use within an organization or company is not restricted. [55] If a copy is given to another organization or individual, then it is a distribution [55].

The second question that arises concerns what happens to service providers who develop a product within an organization. If the copyright is signed over to the client, the license compliance artifacts do not have to be provided. However, the client may want it, and therefore the contract of the project should be checked. Besides, even if there is no agreement regarding license compliance, it is still a topic that should be discussed [56].

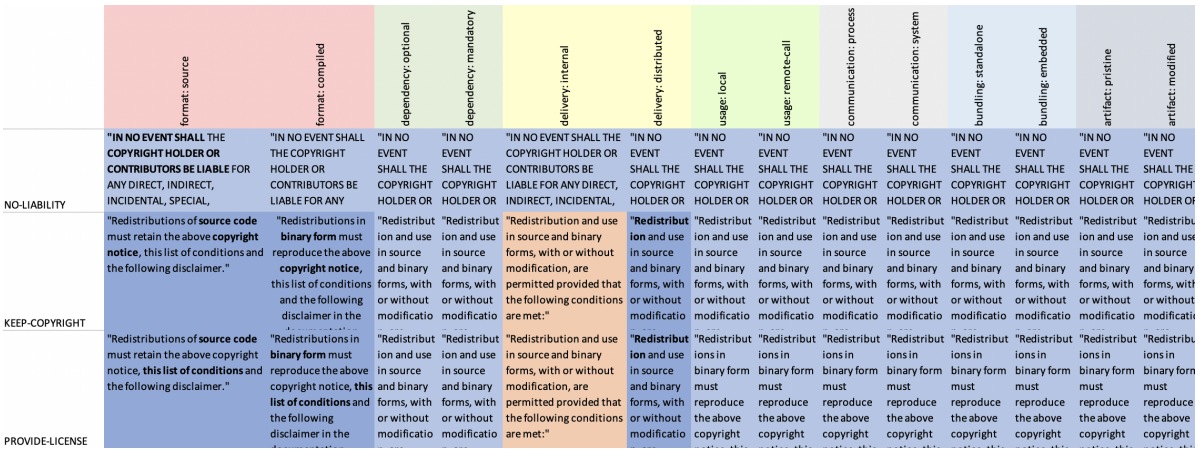


Figure 4.7: Excerpt of BSD-2-Clause License Analysis [45]

- Especially for strong Copyleft licenses, the use type is relevant. A strong copyleft effect is usually a knockout criterion for the use of a component. Considering the license analysis of the CPL-1.0 license, notice that copyleft-strong is an obligation. Strong copyleft means that the user has to provide the source code of the entire product. Internal use is excluded, which means that the internal use of a component under a copyleft license is harmless. However, if the user wants to use and distribute the component, the usage is the decisive factor. If the component is used standalone, the usage can be harmless. However, this also assumes that the product is not dependent on the component. The product must, therefore, be able to function without this component. If this is the case, then the component can be used without the provision of the own work. The additional condition is represented in the license matrix by " &dependency: optional" and in

the data model by the field "requiresAdditionalUseTypes". Referring to the following statement "If the program uses fork and exec to invoke plug-ins, then the plug-ins are separate programs, so the license for the main program makes no requirements for them. So you can use the GPL for a plug-in, and there are no special requirements.If the program dynamically links plug-ins, and they make function calls to each other and share data structures, we believe they form a single program, which must be treated as an extension of both the main program and the plug-ins. This means that combination of the GPL-covered plug-in with the non-free main program would violate the GPL. However, you can resolve that legal problem by adding an exception to your plug-in's license, giving permission to link it with the non-free main program."[55] the additional use type is relevant.

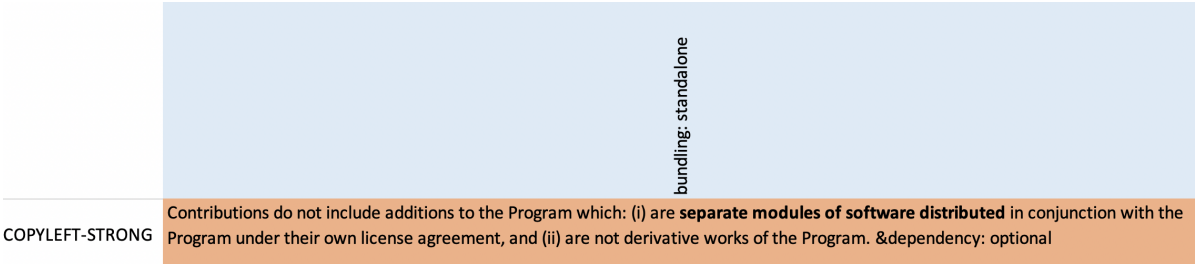


Figure 4.8: Excerpt of CPL-1.0 License Analysis [14]

- In contrast, the licenses LGPL-2.0-only and GPL-2.0-only exclude the complete license terms when the component is used separately. The section "These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the program and can be reasonably considered independent and separate works in themselves, then this license, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the program, the distribution of the whole must be on the terms of this license, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it."[51] indicates that the license and its license terms do not apply if the application is standalone. TThe requirement is that it is independent, i.e., that the product can function without this component[57].

	bundling: standalone
KEEP-COPYRIGHT	These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, **then this License, and its terms, do not apply to those sections** when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.&dependency: optional
PROVIDE-LICENSE	These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.&dependency: optional

Figure 4.9: Excerpt of LGPL-2.0-only License Analysis [51]

- The relevance of the use types "communication: process" and "communication: system" appear in LGPL-2.0-only, LGPL-3.0-only, GPL-2.0-only, and GPL-3.0-only. For LGPL-2.0-only and GPL-3.0-only the section "The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does"[51] states that the act of running the Program is not restricted.

	communication: system
KEEP-COPYRIGHT	"The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does."
PROVIDE-LICENSE	"The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does."
PROVIDE-SOURCE	"The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does."

Figure 4.10: Excerpt of LGPL-2.0-only License Analysis [51]

- Similarly, for the LGPL-3.0-only and GPL-3.0-only licenses, the section "This License explicitly affirms your unlimited permission to run the unmodified program"[17] states

that you are granted unlimited permission to run the unmodified program. That statement contains an additional restriction. It only applies if the component is used unmodified. The additional required use type is, therefore, "&artifact: pristine".

	communication: system
KEEP-COPYRIGHT	"This License explicitly affirms your unlimited permission to run the unmodified Program."&artifact:pristine
PROVIDE-LICENSE	"This License explicitly affirms your unlimited permission to run the unmodified Program." &artifact: pristine
PROVIDE-SOURCE	"This License explicitly affirms your unlimited permission to run the unmodified Program." &artifact: pristine

Figure 4.11: Excerpt of LGPL-3.0-only License Analysis [17]

- Furthermore, the relevance of the use types "usage: local" and "usage: remote-call" is still undefined. They indicate whether a component was called locally or remotely for execution. The licenses LGPL-3.0-only, GPL-3.0-only and AGPL-3.0, use the term convey instead of redistribution[55]. The definition included in each of the licenses is "To 'convey' a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying"[15] states that interaction over a computer network is not conveying. As the licenses state that the obligations apply to convey, the implicit conclusion is that the obligation for the use type "usage: remote-call" can be excluded.

	usage: remote-call
KEEP-COPYRIGHT	"To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying."
PROVIDE-LICENSE	"To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying."
PROVIDE-SOURCE	"To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying."

Figure 4.12: Excerpt of LGPL-3.0-only License Analysis [15]

- After demonstrating the relevance of all use types, the relevance of obligations is considered in the following. The obligation "NO-LIABILITY" is in general included in the licenses by the following disclaimer "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"[44]. With this note, the user acknowledges that no liability is assumed for its use.

	format: source
NO-LIABILITY	**"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."

Figure 4.13: Excerpt of the MIT License Analysis [44]

- The second obligation "KEEP-COPYRIGHT" states that you must retain copyright information. For example, the MIT license contains the following statement: "The above in all copies or copyright notice and this permission notice shall be included substantial portions of the Software"[44].

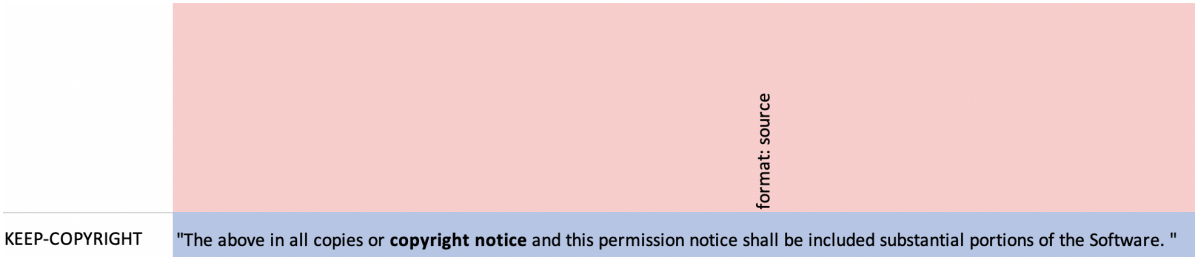


Figure 4.14: Excerpt of the MIT License Analysis [44]

- "PROVIDE-LICENSE" indicates that you must provide the full license text of the component. The excerpt "Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer"[45] from the BSD-2-Clause states that the list of conditions must be provided which corresponds to this obligation.

Figure 4.15: Excerpt of the BSD-2-Clause License Analysis [45]

- "PROVIDE-SOURCE" requires that the full source code of the component is made available. This obligation is only relevant for the use type "format: compiled", since the source code is provided when the component is distributed in source format, and therefore the condition is fulfilled. An example is an excerpt "If You distribute Covered Software in Executable Form then: a. such Covered Software must also be made available in Source Code Form, as described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and..."[12] of the MPL license. Executable form corresponds to the use type "format: compiled".

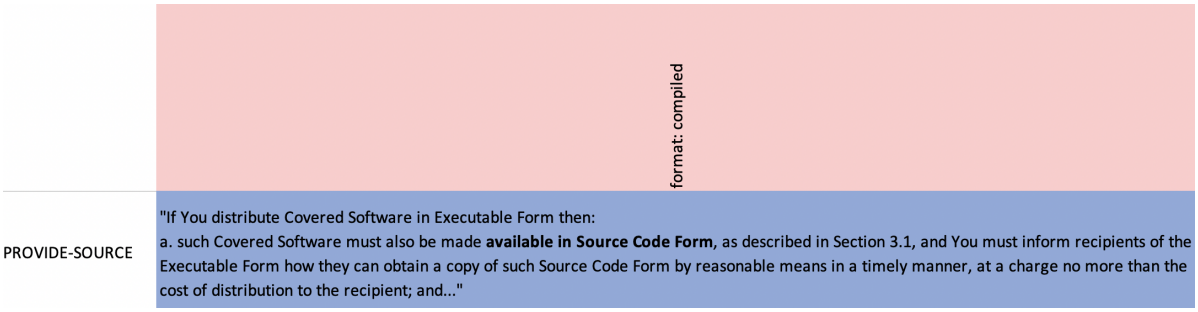


Figure 4.16: Excerpt of the MPL License Analysis [12]

- "ADV-CLAUSE" denotes that the application or its documentation must contain a note about the component and its author. Keywords to find this condition in a license are, for example, "trademarks" or "advertising". For instance, the BSD-4-Clause contains the following statement: "All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the organization"[47].

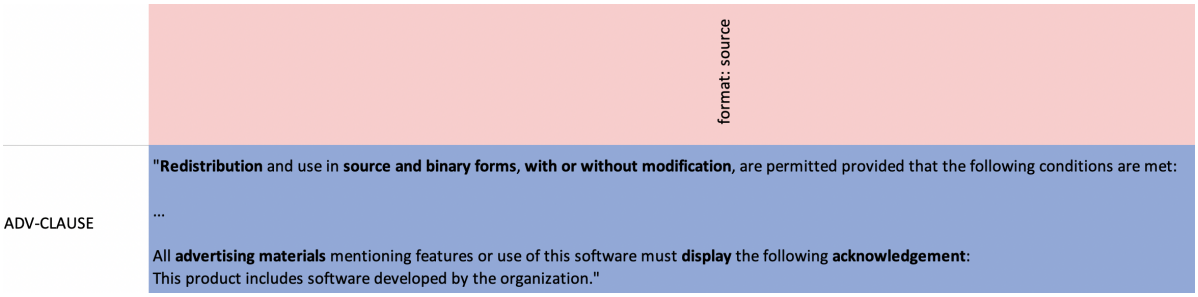


Figure 4.17: Excerpt of the BSD-4-Clause License Analysis [47]

- Furthermore, a statement, for example, included in the BSD-4-Clause is: "Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission."[47]. This statement affirms that the name of the component must be changed when modifying and distributing the component. This condition is represented in the model by the obligation "RENAME".

	format: source
RENAME	**"Redistribution** and use in **source and binary forms, with or without modification**, are permitted provided that the following conditions are met: ... 4. **Neither** the **name of the copyright holder nor the names of its contributors** may be **used** to endorse or promote products derived from this software without specific prior written permission."

Figure 4.18: Excerpt of the BSD-4-Clause License Analysis [47]

- Another obligation, such as "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, excluding those notices that do not pertain to any part of the Derivative Works, 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, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License."[50] of the Apache-2.0 license and which is associated with the obligation "NO-RELICENSE", states that the component may not be relicensed under another license.

	format: source
NO-RELICENSE	"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, excluding those notices that do not pertain to any part of the Derivative Works, 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, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do **not modify the License.** You **may add Your own attribution notices** within **Derivative Works** that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License."

Figure 4.19: Excerpt of the Apache-2.0 License Analysis [50]

- Use in a military context is not permitted by some licenses and is covered by the obligation "CTX-NO-MIL". An example shows the excerpt "You are not listed on the United States Department of Treasury lists of Specially Designated Nationals and Blocked Persons, Specially Designated Terrorists, and Specially Designated Narcotic Traffickers, nor are You listed on the United States Department of Commerce Table of Denial Orders"[53], which refers to the OTN license.

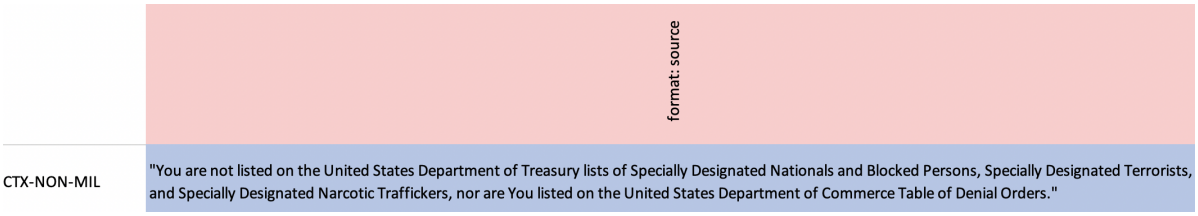


Figure 4.20: Excerpt of the OTN License Analysis [53]

- The OTN is not a common OSS license. Therefore some obligations exist to represent other licenses, which are not OSS licenses but are relevant in practice. In license analysis, there are two such licenses, the OTN, and the SDN. Both licenses contain conditions that do not conform to the OSS definition. Regarding the section "Oracle Employees: Under no circumstances are Oracle Employees authorized to download software for the purpose of distributing it to customers. Oracle products are available to employees for internal use or demonstration purposes only"[53] from the OTN license, the OTN license states that Oracle employees have exclusive rights of use, which contradicts the equality of the OSS definition.

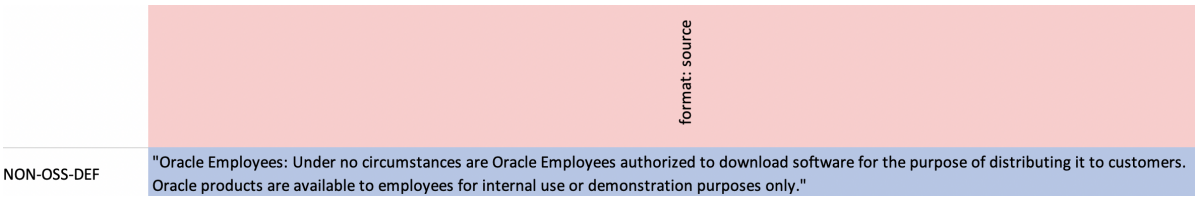


Figure 4.21: Excerpt of the OTN License Analysis [53]

- Further restrictions exist regarding copyleft. Some licenses require that the modified version of the component or even the complete product for which the component is used is made available."COPYLEFT-WEAK" indicates that not the complete product inherits the copyleft license. The LGPL-2.0-only license e.g., covers this case: "You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License."[51].

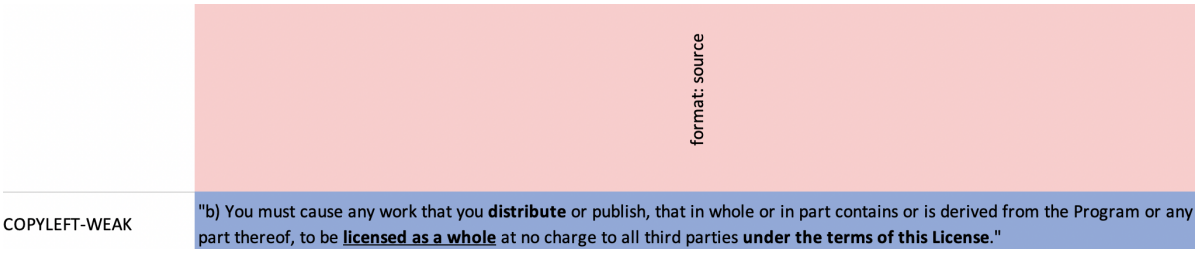


Figure 4.22: Excerpt of the LGPL-2.0-only License Analysis [51]

- "COPYLEFT-STRONG", on the other hand, states that the complete work as a whole must be made available under the license, what for instance can be found in the GPL licenses ("But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it"[52]).

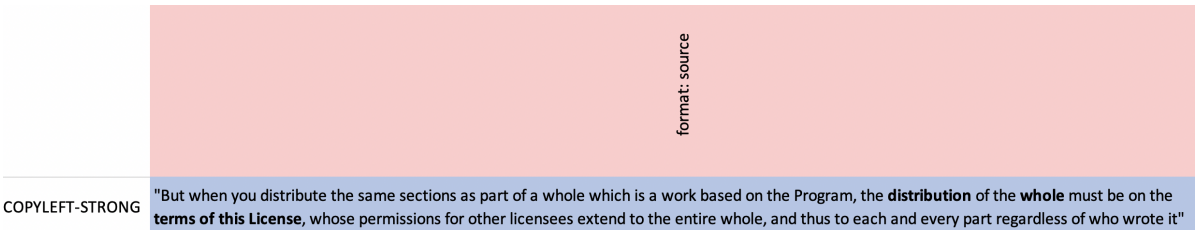


Figure 4.23: Excerpt of the GPL-2.0-only License Analysis [52]

- Finally, an obligation "OTHER" is defined, which represents obligations that are not considered in the modeling. These are mostly contained in copyleft licenses, which you probably are not allowed to use. If the type of use allows the use of such a license, this requires another manual check. An example of this would be that you must include installation instructions when distributing the product, according to the following excerpt of the GPL-3.0-only license: "'Installation Information' for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must be sufficient to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made."[15].

	usage: remote-call
OTHER	""“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made." "Sublicensing is not allowed; section 10 makes it unnecessary." "a) The work must carry prominent notices stating that you modified it, and giving a relevant date."

Figure 4.24: Excerpt of the GPL-3.0-only License Analysis [15]

4.3.3 Graduation of use types and obligations

For calculation, the obligations are assigned values that indicate the overall impact on the product. The range is from one to five, whereby five describes a harmless condition and one a strong one (Figure 4.25). For example, the obligations "NO-LIABILITY", "KEEP-COPYRIGHT", PROVIDE-LICENSE" have the value five. To violate these obligations, the user needs to do something intentionally, such as remove the copyright information.

For obligations with the value three or four, certain conditions must be fulfilled, e.g. by providing the source code. These conditions are not exclusion criteria but must be fulfilled in order to use the component. A commercial product is intended to provide a minimal amount of information so that it is not simply imitated. Therefore, the obligation "ADV-CLAUSE" contains a lower value than, for example, the simple provision of the source code. The obligation "OTHER" gets the average value, as these obligations are not covered, and it requires effort to check whether these conditions can be met.

However, licenses that contain such conditions usually already have obligations that are part of the knockout criteria. Such obligations have the values one or two. These values are assigned to obligations such as copyleft or if used in a commercial environment is prohibited (Figure 4.26). Checking the "OTHER" conditions is only relevant if, e.g., the copyleft condition is excluded by the specific usage type, as can be the case with GPL-2.0-only if the component is used separately and optionally.

Figure 4.26 represents the entire UseTypeObligationCombination matrix. It includes the rating of the obligations as well as a rating of the individual use types for certain obligations. The red marked cells indicate a more harmless use type, whereas blue indicates a more rigorous use type.

Whether the component is used in source or compiled form makes a difference when it comes to the obligations "PROVIDE-SOURCE" AND "ADV-CLAUSE". When using the component in source form, the source code is provided when distributing the component. Also, the source code already contains the advertisement clause, so the user automatically complies with those license terms.

Furthermore, there is a difference between internal and distributed usage. In the case of internal usage, the obligation "NO-LIABILITY" is harmless, since no one within the organization causes someone to be liable. A similar situation applies to other obligations for internal use. Internally, there is usually no complain, which is why the organization is not as compliant with the license conditions as in case of distribution. However, the conditions must be met, but the minor distinction is still necessary. More strict values for the use type "delivery: distributed" are assigned to the obligations "COPYLEFT-STRONG", "NON-OSS-DEF" and "OTHER". Since the core aspect of a copyleft license is the distribution, the value must be stricter than for internal use.

The last two obligations represent potential for trouble, and even if some conditions are not modeled, they must still be met and can cause trouble. If the component is embedded, it looks like the user's own work and the user is partly to blame. Therefore the standalone use of the component is more harmless. The difference is particularly remarkable with "COPYLEFT-STRONG" since when the component is embedded, the complete product must be placed under the license of the component. Furthermore, a distinction is made between using the unmodified or modified component. In this case, the user is partly to blame and cannot hold anyone else liable.

Additionally, the obligation "RENAME" is stricter, since the names must actually be changed after a modification. Furthermore, when re-licensing is forbidden, a modification is stricter, because the user might provide his own changes under different conditions. Also, the "COPYLEFT-STRONG" condition is even stronger after a modification since the modifications must be provided.

Condition Id	DEFCON
NO-LIABILITY	5
KEEP-COPYRIGHT	5
PROVIDE-LICENSE	5
PROVIDE-SOURCE	4
ADV-CLAUSE	3
RENAME	5
NO-RELICENSE	4
CTX-NON-MIL	4
CTX-NON-COM	1
COPYLEFT-STRONG	1
COPYLEFT-WEAK	2
NON-OSS-DEF	2
OTHER	3

Figure 4.25: Obligation Values

	format: source	format: compiled	dependency: optional	dependency: mandatory	usage: local	usage: remote-call	communication: process	communication: system	delivery: internal	delivery: distributed	bundling: standalone	bundling: embedded	artifact: pristine	artifact: modified
NO-LIABILITY	5	5	5	5	5	5	5	5	6	5	6	5	5	6
KEEP-COPYRIGHT	5	5	5	5	5	5	5	5	5	5	5	5	5	5
PROVIDE-LICENSE	5	5	5	5	5	5	5	5	5	5	5	5	5	5
PROVIDE-SOURCE	6	4	4	4	4	4	4	4	5	4	4	4	4	4
RENAME	5	5	5	5	5	5	5	5	6	5	5	5	5	4
ADV-CLAUSE	6	3	3	3	3	3	3	3	5	3	4	3	3	3
NO-RELICENSE	4	4	4	4	4	4	4	4	4	4	4	4	4	3
CTX-NON-MIL	4	4	4	4	4	4	4	4	4	4	4	4	4	4
CTX-NON-COM	1	1	1	1	1	1	1	1	1	1	1	1	1	1
COPYLEFT-WEAK	2	2	2	2	2	2	2	2	2	2	2	2	2	2
COPYLEFT-STRONG	1	1	1	1	1	1	1	1	2	0	1	0	1	0
NON-OSS-DEF	2	2	2	2	2	2	2	2	2	1	2	2	2	2
OTHER	3	3	3	3	3	3	3	3	4	2	3	3	3	3

Figure 4.26: UseTypeObligationCombination

4.4 Roles

Anonymous

An anonymous is a person who gets access to specific information on the system. It can, for example, be a customer who wants to see information about his product.

Admin

An admin is the administrator of the system, and he has the right to read and write every entity of the system.

Architect

An architect is a person who develops a product and checks whether his software is affected by license compliance issues.

Component Expert

A component expert is a person who has know-how about software components. A person with this role is responsible for maintaining components and for approving changes on components. Also, they can be asked for help if someone needs help according to components.

License Expert

A component expert is a person who has know-how about licenses. A person with this role is responsible for maintaining licenses and for approving changes on licenses. Also, they can be asked for help if someone needs help according to licenses.

Manager

A manager is a person from the management who might be interested in product details according to a specific product or in reports about the most used components to decide which vendor to sponsor.

4.5 Process

4.5.1 Business Process Model

The business process model (Figure 4.27) represents the process an architect is going through when checking the components of their products if they comply with the license terms. The user stories contain some additional functionalities of the system, which are not relevant for the process, such as editing the user preferences. Since the admin is not relevant for the license compliance check process and serves only the administration of the system, he is not part of the model. Also, the manager is not part of the compliance process, but can only view the information from the products of his team, so he is not contained in the model as well.

4.5.2 User Stories

User Settings Preferences (#SETTINGS)

- As an Admin / Architect / License Expert / Component Expert / Manager, I can view and edit my user information so that I configure the user interface for my preferences (name, username, roles, OrgUnit, language, theme, rights, and the profile picture).
- As an Admin, I can view and edit user information so that he can pre-configure others' accounts or help others edit their user information.

Administration of System Rights (OrgUnits, User, Rights) (#ORG)

- As an Admin, I can manage the rights, roles, and OrgUnits of any user of the system so that he can pre-configure the system or help others edit their OrgUnit.
- As a Manager, I can view and edit my own OrgUnit so that I can manage it.

Administration of License Model (#LICENSE)

- As a License Expert, I can create and edit licenses so that they are available in advance for components and products.
- As an Architect, I can create and edit a license so that they are ad-hoc available for components and products.

Administration of Components (#COMPONENT)

- As a Component Expert, I can create and edit components so that they are available in advance for products.
- As an Architect, I can create and edit a component so that they are ad-hoc available for components and products.

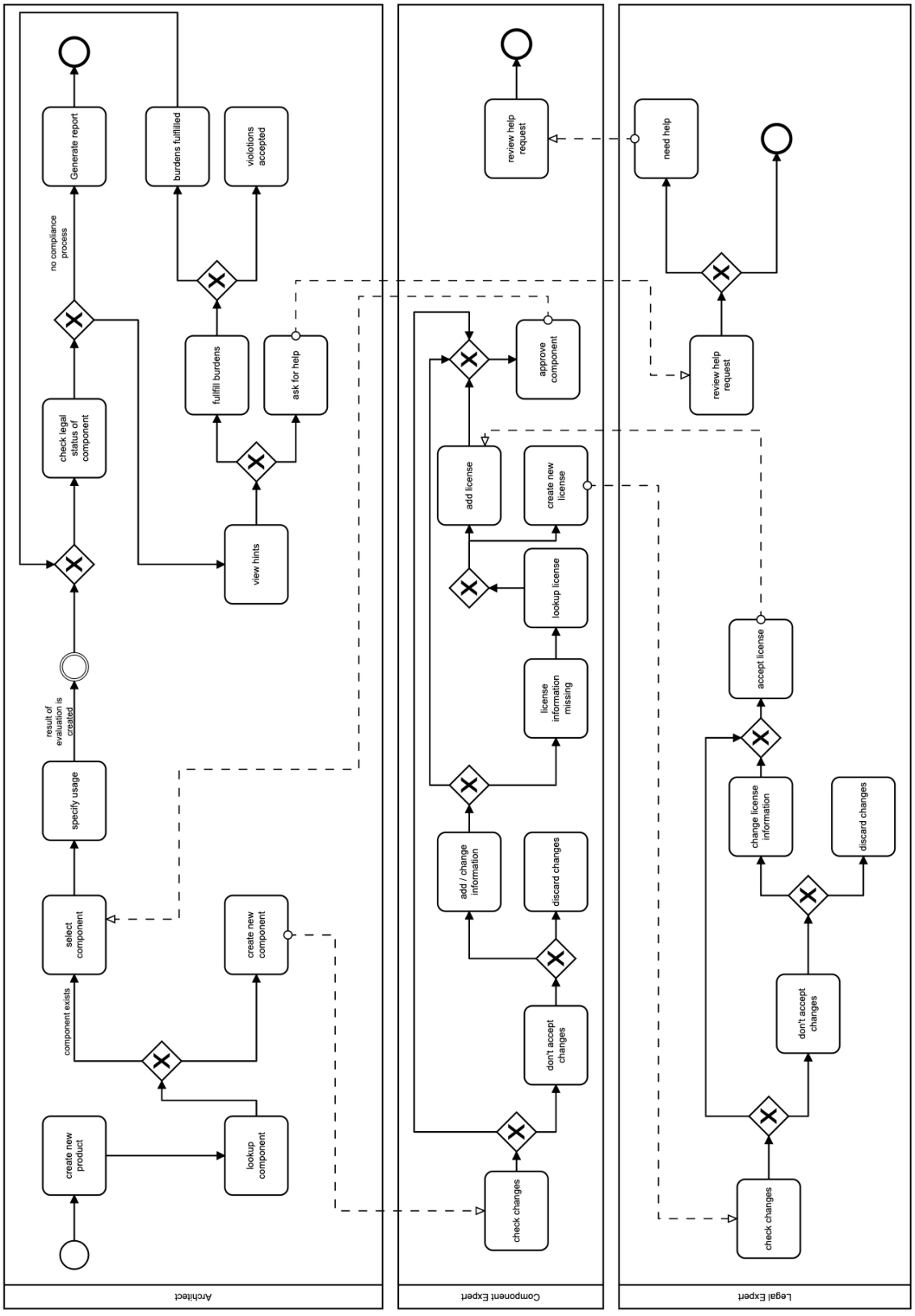


Figure 4.27: Business Process Model

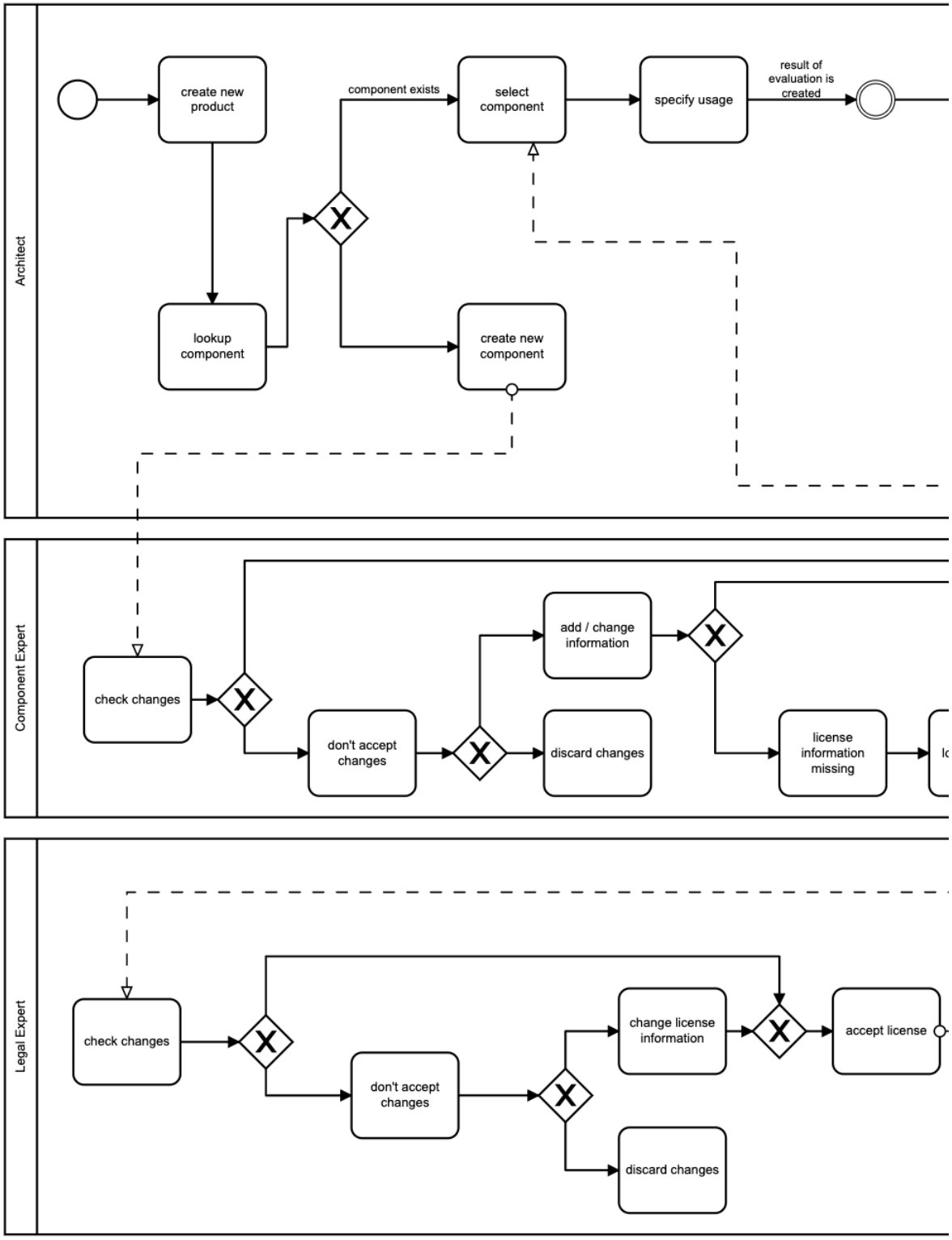


Figure 4.28: Business Process Model - Left part

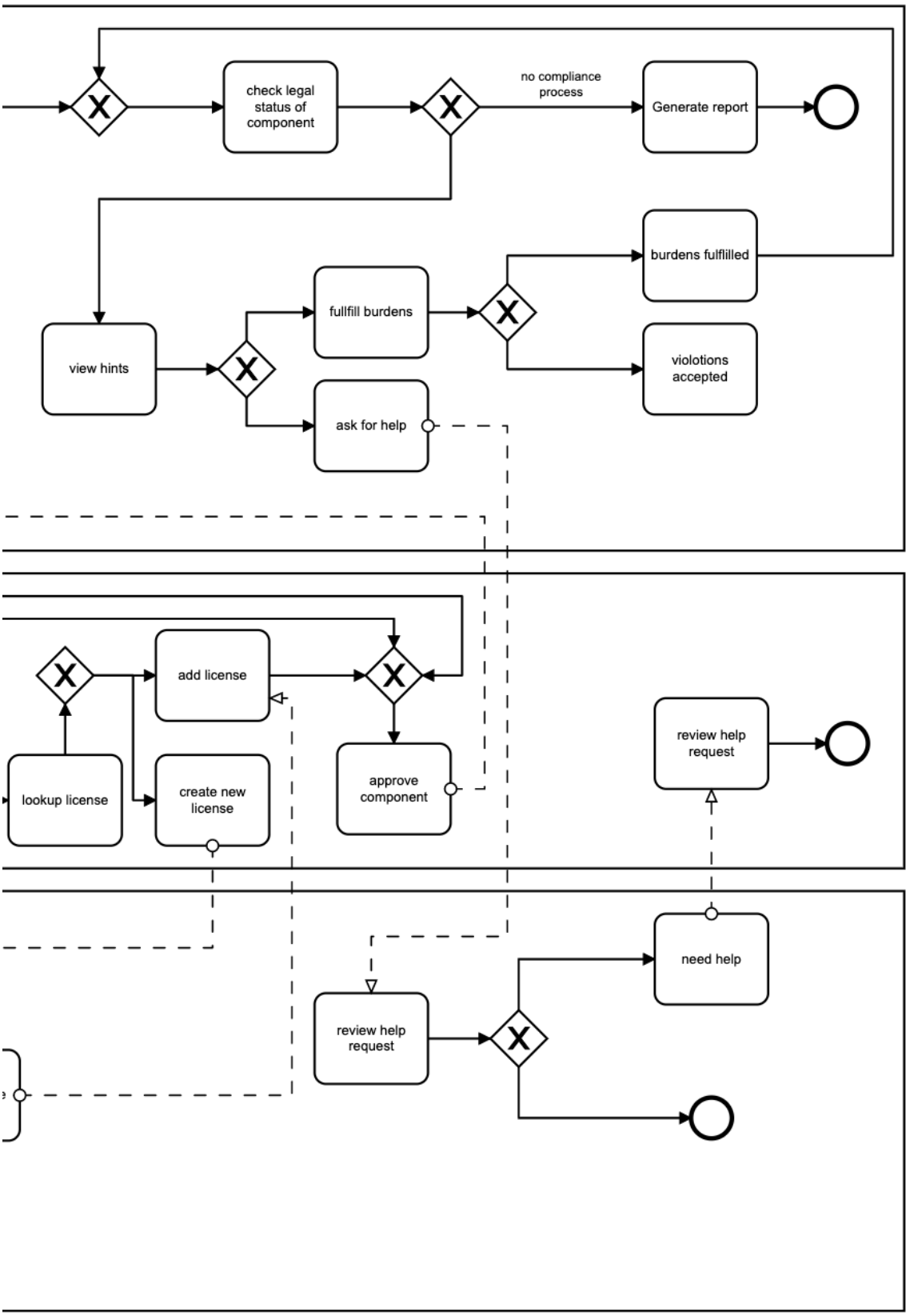


Figure 4.29: Business Process Model - Right part

Administration of Product (#PRODUCT)

- As an Admin, I can view, create, and edit a product of any person (Architect) so that I can help others edit their product.
- As an Architect, I can create and edit a product so that I can capture all product information to calculate a DEFCON level.
- As a Manager, I can view each product in my OrgUnit so that I can review it.

Report Request (#REPORT)

- As a Manager, I can download and view a report of all products within my OrgUnit so that I can overview the legal status of my products.
- As a Architects, I can download and view a report of my own products so that I can have an overview of the legal status of my products, the most frequency of use of components as a basis for making decisions regarding sponsorship and decide which vendor to sponsor and of the security issues of my products.

Request for help (#HELP)

- As an Architect, I can request help regarding the DEFCON level of a product or component so that I get legal or technical support.
- As a License Expert, I get notified when an architect sends a help request so that he can support him.
- As a Component Expert, I get notified when an architect sends a help request so that he can support him.

Approval (#APPROVAL)

- As a License Expert, I can approve license so that they became official.
- As a Component Expert, I can approve license, or platforms changes so that they became official.

4.6 Authorization

The following adjacency matrix (Figure 4.30) describes the authorization of the system, i.e., which rights are assigned to the individual roles. The roles are shown at the top and the user stories on the left, as described above. Each intersection, which consists of a combination of a user role and a use case or its entities, contains an inner matrix (see Table 6.1: Inner Matrix). At the top, you can see whether the entities are your own entities, i.e., entities assigned to the role, or entities of others. On the left, we distinguish between four different cases. The first two indicate whether the user has read and write access. The third case, "#CONFIDENT" tells us if the user needs approval in case of a change. This case can only occur if the user has write access. If the user needs approval for a change, this would be mapped in the fourth case "#UNCONFIDENT/APPROVAL". If we look at the matrix, we notice that the admin has read and write permissions for each entity and is always confident. This means that the admin has all rights by definition.

Table 4.30: Inner Matrix

.	Own	Foreign
#READ		
#WRITE		
#CONFIDENT		
#UNCONFIDENT/APPROVAL		

Settings (#SETTINGS)

- By definition, an Admin always has read and write access and never needs approval. That means he is allowed to access and edit his settings as well as those of other users.
- Users with roles License Expert, Component Expert, Architect, or Manager can access and edit their settings without approval, but not those of other users.
- Users with the role of Anonymous have no settings and therefore have neither read nor write access.

Administration of System Rights (OrgUnits, User, Rights) (#ORG)

- By definition, an Admin always has read and write access and never needs approval. That means he is allowed to access and edit his OrgUnit as well as those of other users.

- Users with the role License Expert, component expert, or architect do not have any OrgUnits and cannot access foreign OrgUnits.
- A Manager has rights access to his own OrgUnit and edit it.
- Users with the role of Anonymous do not belong to any OrgUnit and therefore have neither read nor write access.

Administration of License Model (#LICENSE)

- By definition, an admin always has read and write access and never needs approval. That means he is allowed to access and edit all licenses.
- Users with the role License Expert are the owners of licenses, so there are no foreign ones. They have the right to access and edit them without approval.
- Users with the role Component Expert or Architect do not own any licenses, but foreign licenses in the system can be accessed and edited by these users. For modification, they need approval.
- Users with the role of Manager or Anonymous can view licenses but have no rights to edit them.

Administration of Components (#COMPONENT)

- By definition, an Admin always has read and write access and never needs approval. That means he is allowed to access and edit all components.
- Users with the role Component Expert are the owners of components, so there are no foreign. They have the right to access and edit them without approval.
- Users with the role License Expert or Architect do not own any components, but foreign components in the system can be accessed and edited by these users. For modification, they need approval.
- Users with the role Manager or Anonymous can view components but have no rights to edit them.

Administration of Product (#PRODUCT)

- By definition, an Admin always has read and write access and never needs approval. That means he is allowed to access and edit all products.
- Users with the role of License Expert or Component Expert do not own any products. In case they receive a help request, they can access the products to evaluate the product.
- Architects can access their projects in which they are involved.

- A manager can access the own products as well as those of others within his OrgUnit.
- Users with the Anonymous role can view projects if they have been shared with them.

Report Request (#REPORT)

- By definition, an Admin always has read and write access and never needs approval. That means he is allowed to access and edit all reports. As the system automatically generates a report, write permissions, in this case, are only for deletion.
- Users with the role License Expert or Component Expert do not own any products and, therefore, no reports. In case they receive a help request, they can access the reports to evaluate the associating product.
- Architects can access reports of their projects in which they are involved. As the system automatically generates a report, write permissions, in this case, are only for deletion and needs no approval.
- A manager can access reports of the own products as well as those of others within his OrgUnit.
- Users with the Anonymous role can view reports of projects if they have been shared with them.

Request for help (#HELP)

- By definition, an Admin always has read and write access and never needs approval. That means he is allowed to access and edit all help requests.
- Users with the role License Expert or Component Expert do not own any help requests. However, they can view and edit help requests they get without approval.
- Architects can access their help request and edit them without approval.
- Manager and Anonymous are no actors of the approval process and therefore have no rights.

Approval (#APPROVAL)

- By definition, an Admin always has read and write access and never needs approval. That means he is allowed to access and edit all approvals.
- Users with the role License Expert or Component Expert can access and edit their approvals. A License Expert owns the licenses and the Component Expert, the one for components and platforms. Nevertheless, they cannot see the approval of each other.
- Users with the roles Architects, Manager, or Anonymous are no participants of the approval process.

Platform (#PLATFORM)

- By definition, an Admin always has read and write access and never needs approval. That means he is allowed to access and edit all platforms.
- Users with the role Component Expert are the owners of platforms, so there are no foreign. They have the right to access and edit them without approval.
- Users with the role License Expert or Architect do not own any platforms, but foreign platforms in the system can be accessed and edited by these users. For modification, they need approval.
- Users with the role Manager or Anonymous can view platforms but have no rights to edit them.

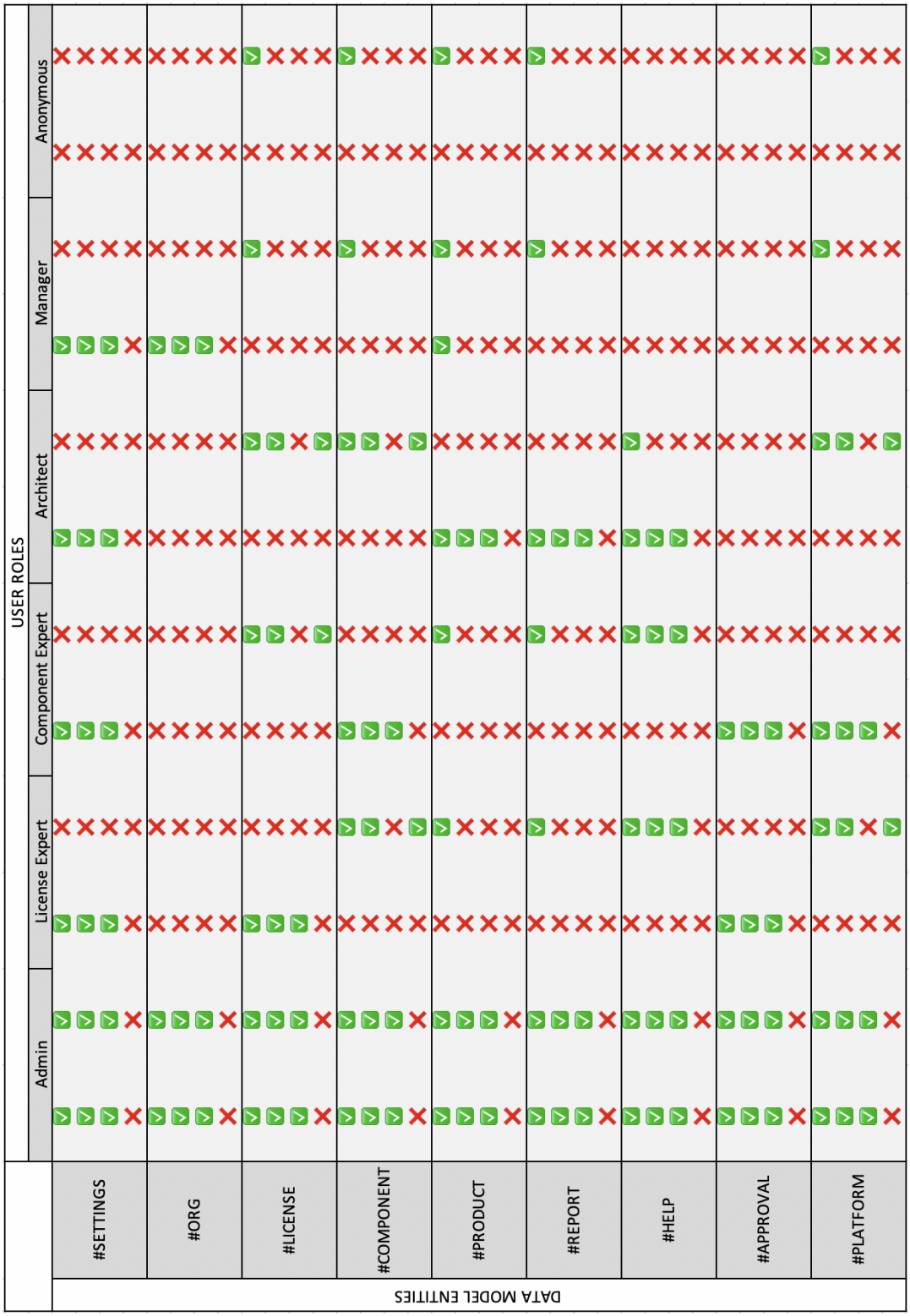


Figure 4.30: Authorization

5 Implementation

5.1 Implementation

The calculation of the algorithm (section 5.2[1]) starts with the component usage on the right side of the data model and works its way through to the left. First, a class called `Evaluation` is created, in which the results of the calculation are stored as an object. The class `Defcon` calculates the legal status of a component usage.

It starts with a product that inherits from a component and has several `ComponentUsages`. A function called `evaluateProduct` iterates over each of these `ComponentUsages` to calculate the legal status. For the calculation a function `calcDefconLevelForComponentUsage` is called for each component usage. This first creates a new evaluation object, in which the results are stored.

Then another method, `calcDefconLevelForComponent`, is executed to calculate the legal status of the respective component. Since a ComponentUsage is associated with a component, which in turn may have transitive dependencies, in a further step, each dependent component is recursively called to calculate its status. That ensures that their transitive dependencies are also taken into account.

The function `calcDefconLevelForComponent` first determines the correct VersionRange of the component to get the license information for calculation. The license information includes the licenses with the respective position for the formula used for the calculation. For each license, the legal status of the component is calculated, including subsequent licenses in case subsequent is true.

The function `calcDefconLevelForLicense` calculates a value for the respective component usage and the corresponding license. For this purpose, an iteration is executed over each license declaration of the respective license. A usage type that is not included in the license declaration will be skipped as they are not relevant for the calculation. In the case of the specified use type that matches the use type of the license declaration, the system first checks what type of obligation is involved.

If it is a "NOT_OBLIGATION_GLOBAL", the value infinite is returned.

If it is a "NOT_OBLIGATION_SINGLE", a check is made whether an additional condition is required. Then, if this is the case, one checks whether this is also fulfilled.

[1]For the presentation of the source code the following OSS component is used: `https://bit.ly/2QQO04J`

Otherwise, case 3 is executed, which checks whether the condition has already been mitigated. If no mitigation exists, the minimum from the value assigned by the UseTypeObligationCombination and the current value is calculated (worst-case).

A fourth case whether the mitigation is partially fulfilled. If this is the case, the value is adjusted by a factor of 2.

The result object evaluation contains the license, the associated conditions, and the license information, which are responsible for the result. The method returns the calculated value. A formula is applied to the four calculated values. Therefore two additional functions are provided.

The first formula `licenseAnd` calculates the minimum from the two calculated values because this means that both license conditions must be met, and therefore, the worst case must be used. The second formula is `licenseOr`. This function evaluates the maximum since only one license has to be complied with, and therefore the more harmless one is chosen.

The result of a component usage evaluation is an object which contains the name of the component, the license, the calculated value called defcon, the dependencies (children), the version, a list of obligations which must be fulfilled

Note: The data is queried from the database by GraphQL [2], which provides the functionality sequelize. Sequelize allows the user to query the relations from the data model in both directions. Thus it is possible to get directly to the mitigation in the algorithm via the UseTypeObligationCombination, although the data model does not provide any relationship in this direction. The relation between Mitigation and ComponentUsage is relevant for the query.

5.2 Algorithm

```
1  class Evaluation {
2      constructor (name, version, useType) {
3          this.name = name
4          this.license = ""
5          this.defcon = Number.POSITIVE_INFINITY
6          this.children = []
7          this.version = version
8          this.useType = useType
9      }
10
11     setClause (clause) {
12         this.clause = clause
13     }
14
```

[2]"GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data."[58]

```
15      setObligation (obligation) {
16          this.obligation = obligation
17      }
18
19      setDependent (component) {
20          this.children.push(component)
21      }
22
23      setDefcon (defcon) {
24          this.defcon = defcon
25      }
26
27      setLicense(license) {
28          this.license = license
29      }
30
31  }
32
33  class Defcon {
34
35      static evaluateProduct(componentUsages, productName) {
36
37          let components = []
38          let defcon = Number.POSITIVE_INFINITY
39          for (let componentUsage of componentUsages) {
40              let ev = this.calcDefconLevelForComponentUsage(componentUsage)
41              components.push(ev)
42              defcon = Math.min(defcon, ev.defcon)
43          }
44
45          return {name: productName, defcon: defcon, children: components}
46      }
47
48      /* calculates the worst case of the defcons of the component and all transitive
            dependencies */
49      static calcDefconLevelForComponentUsage (componentUsage) {
50
51          /* create a new object with information about component */
52          let evaluation = new Evaluation(componentUsage.component.name,
                componentUsage.version, componentUsage.isSpecifiedWith)
53
54          /* calculates defcon of componentUsage (first level) */
55          let defcon = this.calcDefconLevelForComponent(componentUsage.component,
                componentUsage, componentUsage.isSpecifiedWith, evaluation)
56
57          /* calculated dependent components and calculates the worst case */
58          for (const dependentComponent of componentUsage.component.dependsOn) {
59              const evaluationDep = this.calcDefconLevelForComponentUsage(dependentComponent)
60              evaluation.setDependent(evaluationDep)
61              defcon = Math.min(defcon, evaluationDep.defcon)
62          }
63
```

```
64          /* writes the calculated defcon level into the object */
65          evaluation.setDefcon(defcon)
66
67          return evaluation
68      }
69
70      /* evaluate defcons of each component and apply the formula */
71      static calcDefconLevelForComponent (component, componentUsage, useTypes, evaluation) {
72          /* find range, in which the used component lies */
73          const version = componentUsage.version.arithmetic
74          const componentVersionRange = component.definedRange.find((range) => {
75              return range.from.arithmetic <= version && range.to != undefined && version <=
                    range.to.arithmetic
76          })
77
78          /* calculates the defcons of each license including subsequent licenses (dual
                licensing) which are put into the formula */
79          let defcons = [ null, null, null, null ]
80          for (const licenseReference of componentVersionRange.licenseReferences) {
81              if (licenseReference.subsequent) {
82                  /* put subsequent licenses into a list (e.g. GPLv3 or higher) and calculate
                        the defcons */
83                  let licensesS = [ licenseReference.license ]
84                  for (const license = licenseReference.license; license.subsequent != null;
                        license = license.subsequent)
85                      licensesS.push(license.subsequent)
86                  let defcon = licensesS.map((license) => this.calcDefconLevelForLicense(
                        license, usageTypes, evaluation))
87
88                  /* aggregate calculated defcons to the best option with least burdens */
89                  defcons[licenseReference.position - 1] = defconsS.reduce((a,b) => Math.max(
                        a,b), Number.NEGATIVE_INFINITY)
90              }
91              else
92                  defcons[licenseReference.position - 1] = this.calcDefconLevelForLicense(
                        licenseReference.license, useTypes, evaluation)
93          }
94
95          /* evaluate formula for multi-licensing (X1 AND X2) OR (X3 AND X4) for multi
                licensing */
96          let defcon = this.licenseOr(
97              this.licenseAnd(defcons[0], defcons[1]),
98              this.licenseAnd(defcons[2], defcons[3])
99          )
100
101         return defcon
102     }
103
104     /* calculates the defcon for a specific license */
105     static calcDefconLevelForLicense(license, usageTypes, evaluation) {
106         /* special case for base classes */
107         if (license.basedOn != null)
```

```
108             license = license.basedOn
109
110         /* start with maximum defcon level */
111         let defcon = Number.POSITIVE_INFINITY
112
113         /* bring license declarations in matrix structur, ordered into a map (obligations,
                [licenseDeclaration]) */
114         let matrix = new Map()
115         for (const licenseDeclaration of license.licenseDeclarations) {
116             /* skip non-relevant usetypes */
117             if (!usageTypes.includes(
                        licenseDeclaration.belongsToUseTypeObligationCombination.belongsToUseType.name
                        ))
118                 continue
119
120             /* sort declarations into obligation-keyed matrix row */
121             const obligation =
                        licenseDeclaration.belongsToUseTypeObligationCombination.belongsToObligation.name

122             let licenseDeclarations = matrix.get(obligation)
123             if (licenseDeclarations == undefined)
124                 licenseDeclarations = []
125             licenseDeclarations.push(licenseDeclaration)
126             matrix.set(obligation, licenseDeclarations)
127         }
128
129         /* iterative over matrix row wise (obligations) */
130         let burdens = []
131         let clauses = []
132         let prev_defcon = 0
133         loop:
134         for (const obligation of matrix.keys()) {
135             prev_defcon = defcon
136             /* iterate over matrix column wise (use types) */
137             for (const licenseDeclaration of matrix.get(obligation)) {
138                 /* CASE 1: stop processing of entire matrix */
139                 if (licenseDeclaration.obligate == "NOT_OBLIGATION_GLOBAL") {
140                     if (licenseDeclaration.requireAdditionalUseType == null ||
141                         (licenseDeclaration.requireAdditionalUseType != null &&
142                         usageTypes.includes(licenseDeclaration.requireAdditionalUseType.name
                                )))
143                         return Number.POSITIVE_INFINITY
144                 }
145
146                 /* CASE 2: stop processing of obligation line (this obligation) */
147                 if (licenseDeclaration.obligate == "NOT_OBLIGATION_SINGLE"){
148                     if (licenseDeclaration.requireAdditionalUseType == null ||
149                         (licenseDeclaration.requireAdditionalUseType != null &&
150                         usageTypes.includes(licenseDeclaration.requireAdditionalUseType.name
                                )))
151                         defcon = prev_defcon
152                         continue loop
```

```
153                 }
154
155                 /* CASE 3: usetypes already sorted above, not in case it has been already
                        mitigated */
156                 if (licenseDeclaration.belongsToUseTypeObligationCombination.hasMitigation
                        == undefined) {
157                     defcon = Math.min(defcon,
                            licenseDeclaration.belongsToUseTypeObligationCombination.defconLevel
                            )
158
159                     if (!burdens.includes(licenseDeclaration
160                     .belongsToUseTypeObligationCombinatio
161                     .belongsToObligation.description))
162                         burdens.push(licenseDeclaration
163                         .belongsToUseTypeObligationCombination
164                         .belongsToObligation.description)
165
166                     let clause = {}
167                     clause.declaration =
                            licenseDeclaration.belongsToUseTypeObligationCombination
168                     clause.excerpt = licenseDeclaration.excerpt
169                     clauses.push(clause)
170                 }
171
172                 /* CASE 4: in case it has been partially mitigated */
173                 else if (
                        licenseDeclaration.belongsToUseTypeObligationCombination.hasMitigation
                        != null &&
174                     licenseDeclaration.belongsToUseTypeObligationCombination.hasMitigation.partially
                            ) {
175                     defcon = Math.min(defcon,
                            licenseDeclaration.belongsToUseTypeObligationCombination.defconLevel
                             + 2)
176                 }
177
178             }
179         }
180
181         evaluation.setLicense(license)
182         evaluation.setObligation(burdens)
183         evaluation.setClause(clauses)
184         return defcon
185     }
186
187     /* special and from our formula (e.g. MIT and additional conditions) */
188     static licenseAnd(x, y) {
189         if (x == null && y != null) return y
190         else if (x != null && y == null) return x
191         else if (x != null && x != null) return Math.min(x, y)
192         else
193             return null
194     }
```

```
195
196     /* special or from our formula (dual licensing) */
197     static licenseOr(x, y) {
198         if (x == null && y != null) return y
199         else if (x != null && y == null) return x
200         else
201             return Math.max(x, y)
202     }
203 }
204
205 module.exports = Defcon
```

5.3 Visualization

A TreeMap[3] of the D3.js library is used to process further the data returned by the algorithm. The following example shows how the treemap is interpreted and used: The product "MyProduct" consists of two OSS components. The data of the components and their dependencies are derived from the original information of Spring Boot[59] and Bootstrap Business Test [60], but does not precisely match the original.

A standard useType instance is used for each component usage of this component this example:

- ☐ format: source
- ☒ format: compiled
- ☐ dependency: optional
- ☒ dependency: mandatory
- ☐ delivery: internal
- ☒ delivery: distributed
- ☒ usage: local
- ☐ usage: remote-call
- ☐ communication: process
- ☒ communication: system
- ☒ bundling: standalone
- ☐ bundling: embedded
- ☐ artifact: pristine
- ☒ artifact: modified

Bootstrap Business Test Tree (MIT)

- Wiremock (Apache-2.0)
  - jackson-annotations (Apache-2.0)
  - jackson-databind (Apache-2.0)
  - slf4j-api (MIT)
- Loremipsum (MIT)

[3]The following link shows the link to the OSS component used to visualize the output: `https://bit.ly/3hTQzPy`

  - junit (EPL-1.0)
    - hamcrest-core (BSD-3-Clause)
- jetty-server (Apache-2.0, EPL-1.0)
  - jetty-xml (Apache-2.0)
  - jetty-util-ajax (Apache-2.0)
  - jetty-slf4j-impl (Apache-2.0)
  - jetty-test-helper (Apache-2.0)
- HSQLDB (HSQLDB License - based on BSD-3-Clause)
- junit-jupiter-engine (EPL-1.0)
- junit-jupiter-params (EPL-1.0)
- bootstrap-core (MIT)
  - jackson-annotations (Apache-2.0)
  - jackson-databind (Apache-2.0)
  - jackson-jaxrs-json-provider (Apache-2.0)
  - jsoup (MIT)
    - gson (Apache-2.0)
      - junit (EPL-1.0)
        - hamcrest-core (BSD-3-Clause)
    - junit (EPL-1.0)
      - hamcrest-core (BSD-3-Clause)
    - jetty-server (Apache-2.0)
    - jetty-servlet (Apache-2.0)
- bootstrap-launcher (MIT)
- mockito-core (MIT)
- spring-test (Apache-2.0)

Spring Boot Tree (Apache-2.0)

- spring-core (Apache-2.0)
  - spring-icl (Apache-2.0)
  - spring-context (Apache-2.0)
    - spring-aop (Apache-2.0)

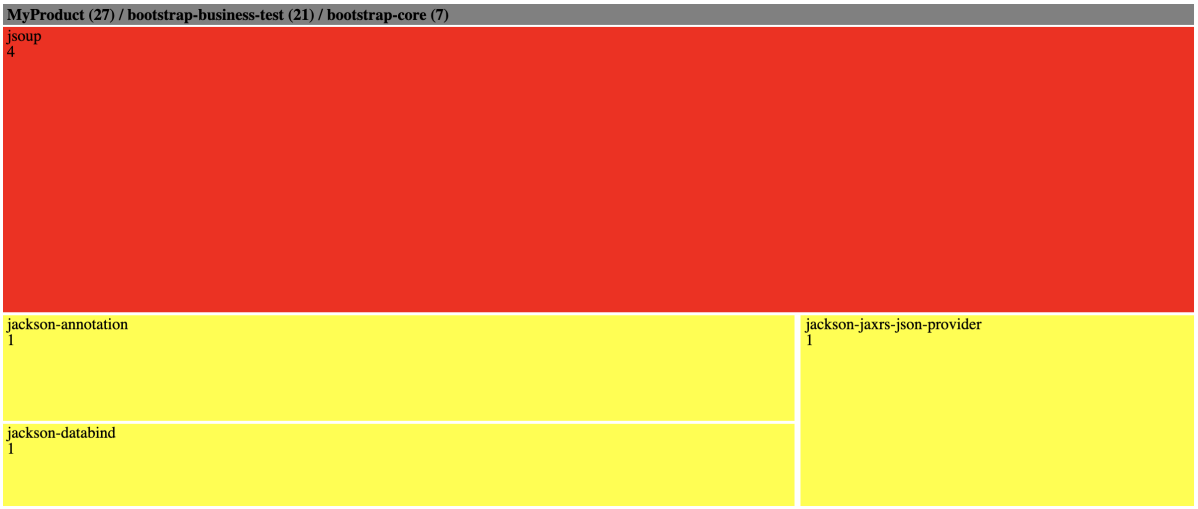
        - spring-beans (Apache-2.0)
            - spring-core (Apache-2.0)
        - spring-core (Apache-2.0)
    * spring-beans (Apache-2.0)
    * spring-core (Apache-2.0)
    * spring-expression (Apache-2.0)
        - spring-core (Apache-2.0)

The above example shows that even the use of two components contains a large number of components, all of which must be checked, and this example does not contain all transitive dependencies of the original components. Executing the algorithm with the two components and the standard use types shown above will result in a large object. The following treemap visualizes the output:

At the first level, the product consists of two components, each represented as a rectangle. The calculated value of each component usage, which ranges from zero to six, is classified into three categories that define the colors of the rectangles. Red rectangles represent component usages with values between zero and two. It indicates that the component should not be used because the license terms cannot be met. Component usages with values three and four are displayed in yellow. Burdens must be fulfilled to comply with the license terms. Green means that all license conditions are met. It contains values higher than four. The text in each rectangle presents the name, and the number below it the number of components contained. The grey rectangle indicates the current path and is used for navigation. Looking into a lower layer, clicking on this rectangle will navigate the user back to the parent. Inspecting the dependencies is useful, for example, to understand the cause of a problem. Considering the Bootstrap Business Test component, it does not cause a problem itself, as it is

licensed under the MIT license. Therefore, the conclusion is that the problem is related to a direct or transitively dependent component on the component. In order to determine the location of the problem, the tree of the component itself is examined.

The grey rectangle shows the path. It indicates that the tree of the Bootstrap Business Test component is displayed. The colors identify which components are causing problems. Observing the component bootstrap-core, which is licensed under the Apache-2.0 license, the cause of the problem can be investigated further by clicking on the component.



The following two graphs represent the view down to the last child node. As the next two graphics show, the component JUnit is one of the problem causing components. It stays under the EPL-1.0 license, which has a strong copyleft effect. This case shows how important

it is to specify the usage. If a component is only used for testing, the architect still needs to comply with the license terms. However, JUnit is only used for testing, and, therefore, will not be delivered to the customer. Since a standard use type is used, which did not specify that the component is only used internally, the component is causing a problem. But, if the usage would be specified for each component, then JUnit would not cause any trouble, since for internal use, the EPL-1.0 is harmless.

5.4 Mockups

The following graphics represent mockups created with the tool Adobe XD. These serve as a basis for the implementation and provide the view of the end-users in the system. The goal is to provide the information and functionalities in a user-friendly and easy way for the end-user.

Figure 5.1: Product View

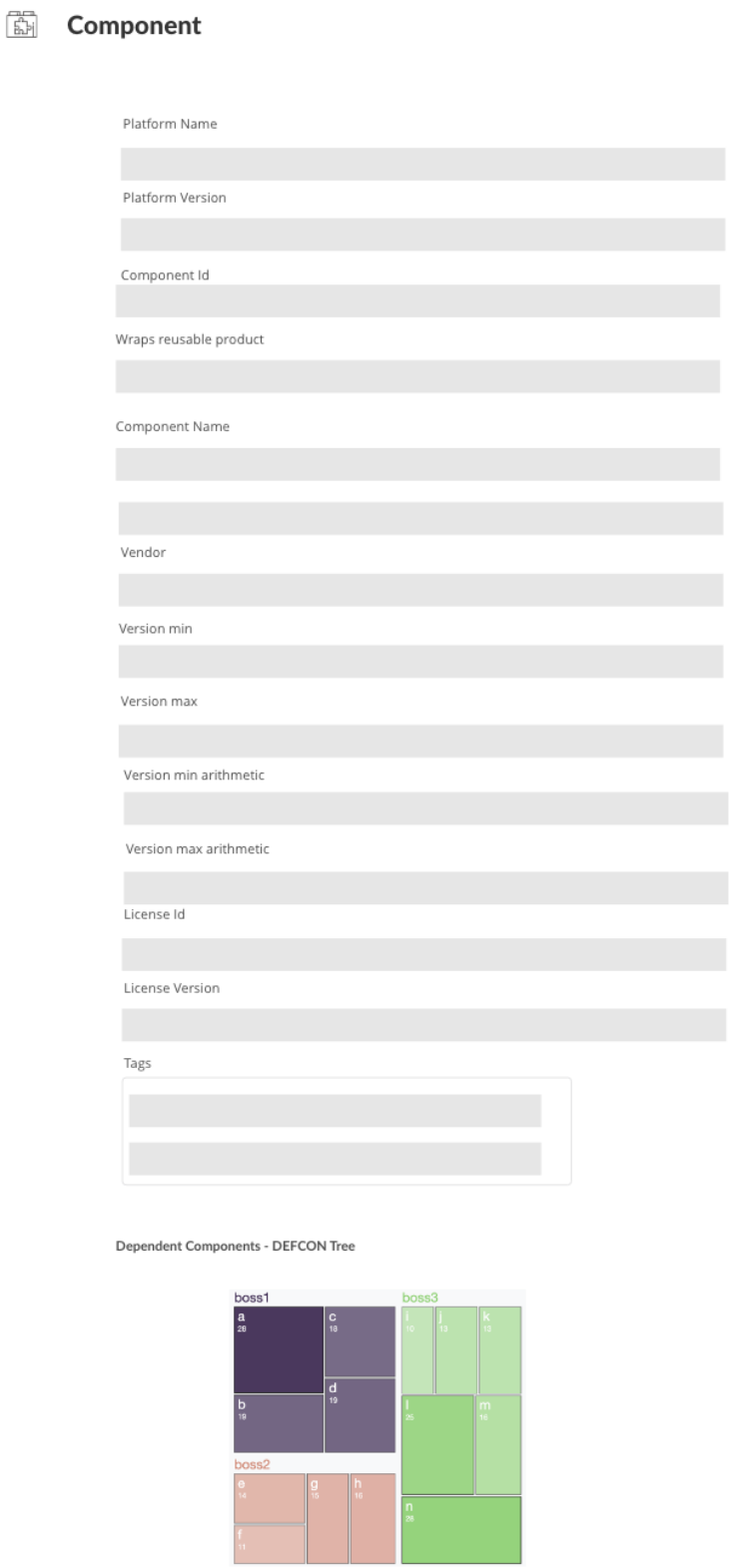


Figure 5.2: Component View

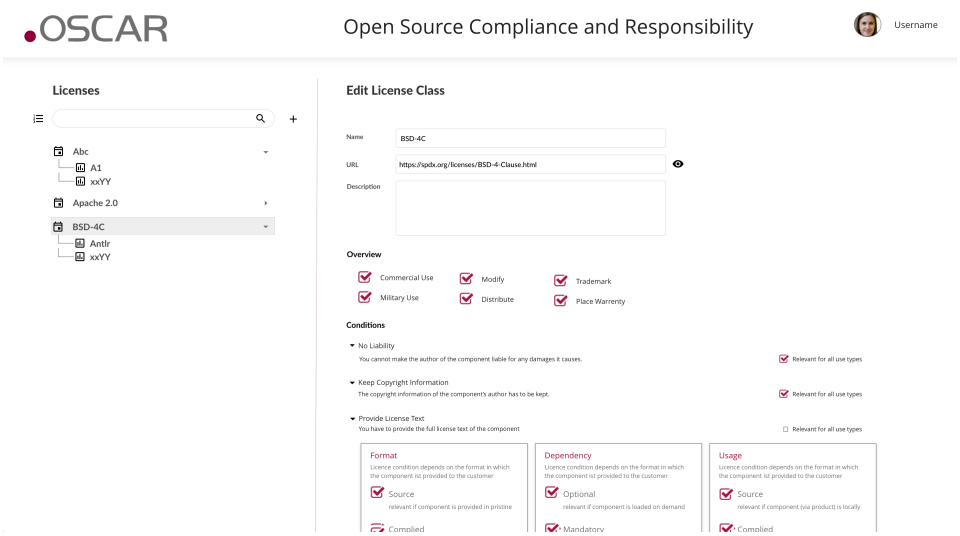


Figure 5.3: License Overview

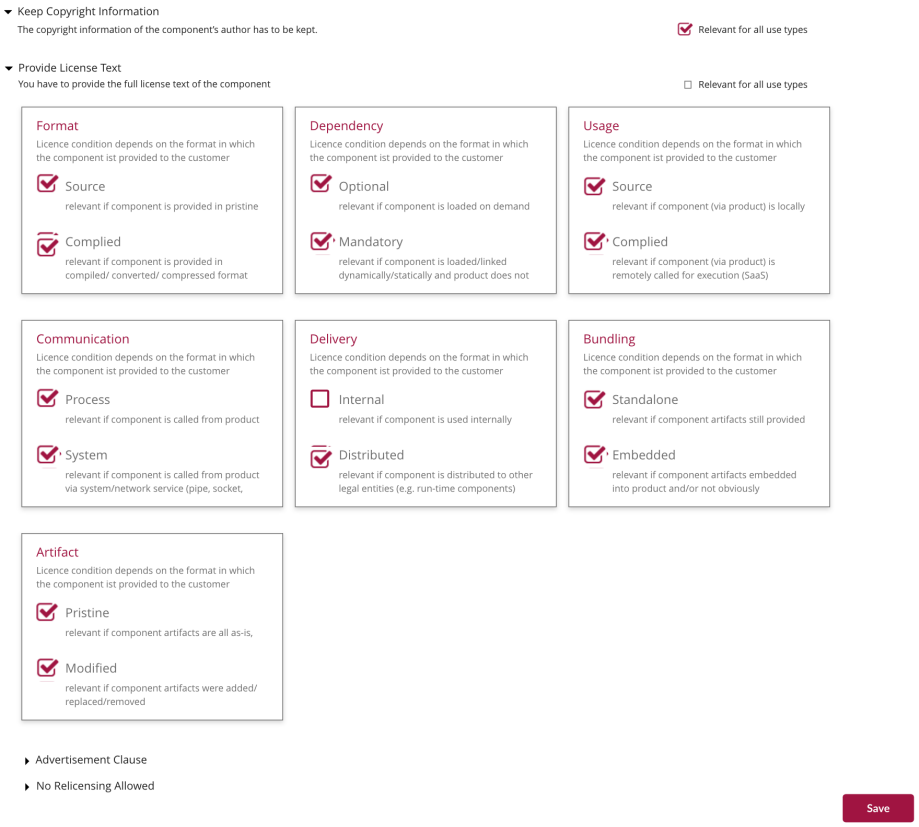


Figure 5.4: License Overview - Part two

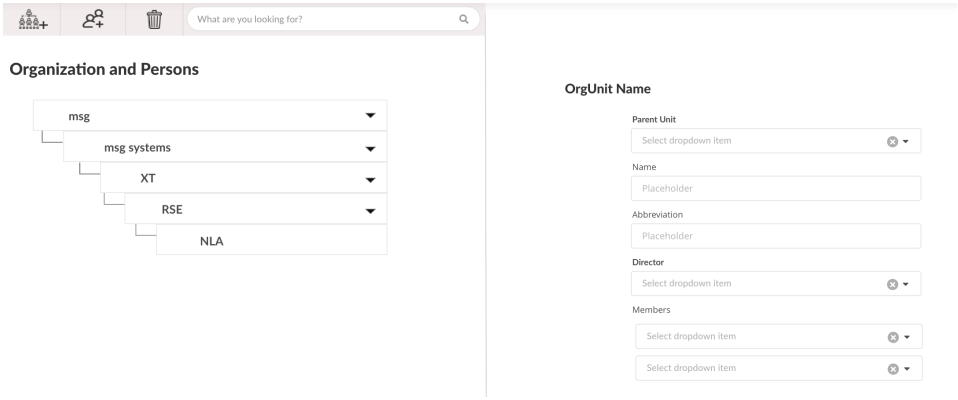


Figure 5.5: Organization and Persons Overview - Edit OrgUnit

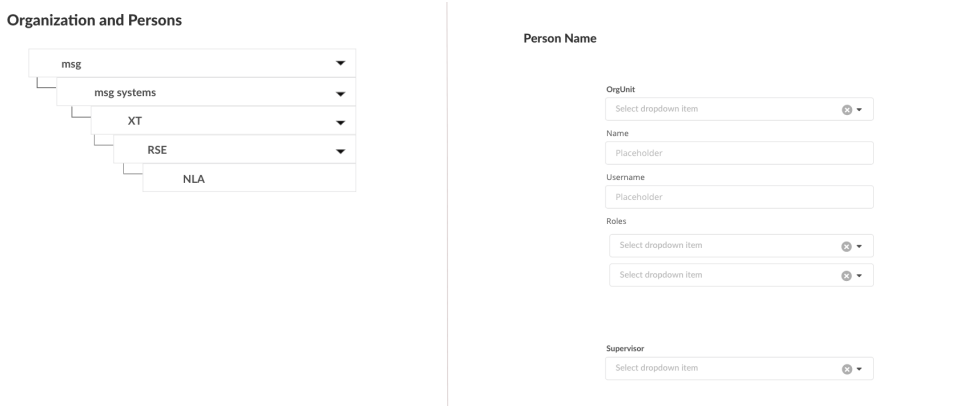


Figure 5.6: Organization and Persons Overview - Edit Person Details

6 Evaluation

6.1 Iterative Process

The support within the company ensured continuous evaluation. We regularly discussed the status and changes through regular status meetings, which took place every one to two weeks. The development team consists of one architect and four master students of information systems. Additionally, my supervisor, Dr. Ralf S. Engelschall, attended. We discussed in each iteration if the new ideas and changes are technically feasible.

Furthermore, the architect was able to check the technical correctness from his experience and gave us suggestions for improving the data model, for example, to avoid technical duplicates. One example is the avoidance of duplicates in licenses. Originally, licenses were assigned to a license class, which were defined by license declarations. This solution did result in duplicates for licenses that represent both an association class and a license itself. An example is the license MIT. MIT is a license on which some licenses are based, e.g., the JSON License, which has the same license terms as MIT but a different name.

Also, MIT is a frequently used license, which can be used for a component. To avoid duplicates, we have adjusted our modeling so that a license only has to be created once. It can have a single parent on which a license is based (e.g., JSON license is based on MIT), or it can already be a license on which other licenses are based, and it has license declarations.

Another learning is that dual licensing is a bit more complicated than we thought. Initially, we assumed that there could be multiple licenses and that the architect could choose the best option with the least number of conditions to fulfill. However, it is a bit more complicated and often results in complex boolean expressions. For example, it is common to specify a license and combine it with an "AND" operator. This insight gave us the idea to introduce our formula. It took several iterations until it was both technically correct and technically feasible.

Additionally, to the regular consultation within the development team, there was also a regular exchange with the User Experience Team. The regular exchange allowed us to check the professionalism and consider whether the ideas can be meaningfully and user-friendly reflected on the user interface.

6.2 Interviews

In order to evaluate the suggested solution, nine interviews were conducted with five architects, two product managers, one legal expert from the legal department of msg, and one external interviewee responsible for information security in an insurance company.

Interview 1	Legal expert of msg Legal Department.
Interview 2	Project Manager who is currently confronted with license compliance from a customer of a big automotive manufacturer.
Interview 3	Architect who is currently confronted with license compliance from a customer of a big automotive manufacturer.
Interview 4	Architect who is currently confronted with license compliance from a customer of a big automotive manufacturer.
Interview 5	Architect who usually develops individual software for the customer.
Interview 6	Architect who develops both product and individual software.
Interview 7	Architect who develops individual software for the automotive sector.
Interview 8	Architect who is also an OSS expert, including knowledge about licenses.
Interview 9	The interviewee is responsible for information security process in an insurance company.

Table 6.1: Interviews to evaluate the solution

Questions

The following questions were prepared before the survey. However, the survey also considered the respondents' interests, and based on their answers, further questions were asked.

1. What is your role in projects?
2. How educated are you on OSS?
3. How do you choose the components for your current project (or previous projects)? Are you responsible for the compliance check?

   — Present solution —

4. (After presenting license overview mockup) Is the license overview useful? How would it benefit your work?
5. How appropriate is the process compared with your previous process? What are the advantages or disadvantages?

6. How could the tool benefit your work?
7. (After presenting visualization) What do you think about the visualization? Is it helpful? Do you have any suggestions for improvement?
8. Are there other aspects that are missing and would encourage you more to use the tool?
9. Do you have any further comments?

Results

The following table summarizes the most frequent statements from the interviews and indicates whether the respective interviewee made this statement.

		Interviews								
	Statement	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Positive Feedback	Central tool with relevant information	✓	✓	✓	✓	✓	✓	✓	✓	✓
	License Overview	✓	✓	✓	✓	✓	✓	✓	✓	✗
	Treemap Visualisation	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Reports	✓	✓	✓	✓	✓	✓	✓	✓	✓
Further Desired Features	Detailed process description	✗	✓	✓	✓	✓	✗	✗	✓	✓
	Automatization	✗	✓	✓	✓	✓	✓	✓	✗	✓
	Notification on Security Issues	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Notification on Component Updates	✓	✓	✓	✓	✓	✓	✓	✓	✗
	Configuration Options for Reports	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 6.2: Overview of evaluation results

- **Interview 1**

  The legal experts at msg are usually not responsible for verifying licenses, and the interview demonstrated that this task is also a challenge for them. Generally, the department itself is responsible for the review and is more familiar with licenses than the legal department. Therefore, the tool would provide significant relief for him since there is currently no defined process, and he has to search for information on the Internet. The tool's information would thus save him the search, and by displaying the license excerpts, they can help the requesting party from a legal perspective.

  The Use Types' gradation seems to make sense to him, but one has to be careful not to build an internal tool and integrate it into a distributed product. However, this is prevented by specifying use types for each component usage and by the check of transitive dependencies. He also considers the visualization component as useful, and there are no suggestions for improvement.

- **Interview 2**

  The project manager realized the license compliance check challenge when confronted with the team by a customer. He is not responsible for selecting the components. That is the task of the architects and includes making sure to comply with the license terms. A detailed description of the process is essential, including defining the restrictions and defining how to deal with them. An example is to answer the question on which granularity authors must be indicated.

  The visualization seems to make sense but would have to be agreed with the architects. He thinks an overview is good but is not sure whether there would have to be a visualization for it. A report with information about which components are used and which licenses they have is beneficial. However, the report should be in a form that can be processed further, since it usually has to be adapted to the needs of the customer.

- **Interview 3 + 4**

  The two architects work together as part of an agile team and were recently commissioned by a client from the automotive industry to review the licenses. Both are not very familiar with licenses. The licenses were partially read, but the obligations are still unclear. However, the licenses' selection and compliance with the license conditions are part of their job. The license overview, which summarizes the information shortly and understandably, would help them for this purpose.

  The excerpts from the licenses should not be missing in the license overview, as the customer requests these, and thus a proof of the displayed information is available. Automation and integration into the build process are significant for the team because the use cases change very frequently, and the initial effort would be too high. The use types could be partially automated, for example, by reading information from the pom.xml. Only as much as necessary should be filled in manually. The rest should be automated. The manual input could be simplified by a list of default use types, which can be edited for each component because usually, 80-90% of the components have the same use type.

  Furthermore, a defined process is desired, which also specifies the exact steps so that the user does not have to check the Internet for further information. For example, since there is no specification on how to provide the license, they wish to have positive and negative examples for orientation. The report is also beneficial but should be provided in a form that can be further processed. Optionally, it should also contain the license excerpts, which are part of the obligations. The conclusion was that a reasonable basis was established, but that automation is desirable for the next step. For the future, they desire to get notified of security issues and security updates.

- **Interview 5**

  The interviewee is an architect who usually develops individual software for a client. He is not familiar with the licenses and does not check whether he complies with the license terms. He assumes that the use is permitted if the component has already been built into other products. There was positive feedback for the creation of a common database and that it should become a web portal and not Excel. The tool would help him to make sure that the license conditions were met.

  He does not need the license excerpts in the license overview because he does not understand them and only wants a simple summary that makes the process as easy as possible. For the overview, he would also like to have a traffic light that shows him directly what is critical, where there is still something to consider, and what is not critical. He finds the visualization helpful to get an overview.

  Automation makes no sense for developing a product because the selection is made before integration, and the components are not yet included in the pom.xml. However, automation could be used for existing products in order to check afterward whether the license conditions have been met. He also considers the report to be beneficial.

  Furthermore, he finds it helpful to be able to mitigate conditions after they have been met. In addition, the help request is useful, which connects him directly with a responsible contact person if he needs help. Generally, he desires a well-explained process where he is told what steps to take. The use type group "communication" is inappropriate because it reminds him of communicating people.

  Alternatively, he suggests "execution type". The word execution should be used in any case. For the future, he would like to have a report every 2-4 weeks, which points out component updates and security issues. This report should make significant statements on these issues so that competence can be shown to the customer.

- **Interview 6**

  The interviewee is an architect who develops both individual and product software. He is very familiar with licenses and has a personal interest in adhering to the license terms, as he appreciates the added value of OSS. Despite his good knowledge about OSS licenses, the license overview helps him, as he can use it as a look-up guide. The associated license excerpts are relevant for him, as he likes to question information, as this information is relevant for making decisions, and therefore he can better justify his decision and disclaim the source of his information.

The report offers an added value. He would use it primarily in pdf form, but would optionally wish for a format that can be further processed. The visualization is attractive to him. It is informative, and he enjoys it. However, the number of components is confusing, as he does not know whether it is the number of components or the number of critical components with licensing problems. In order to have a better overview of all critical components, he would like to have another view that shows all components nested on one screen. For the future, he desires automation and notifications on security issues.

- **Interview 7**

  The interviewee is an architect developing individual software in the automotive sector. In the current process, the architects create a blueprint containing components that are allowed to be used. In the selection process, components with the licenses MIT and Apache-2.0 are usually chosen. To the standard licenses, he is familiar. An overview of the license information could help him to select components. He does not need the license excerpts. However, it would have to be ensured that the information is correct since he relies on it to make decisions. No customer has been interested in a report so far, but it could be made available to the customer if it is generated automatically. He is critical of the tool's maintenance effort, as this is a significant expense. The license modeling will not frequently change, because, in his opinion, the analyzed licenses represent 99% of all required licenses. However, the component information could be read in automatically in the future. Furthermore, he would like to see build process integration and notifications on security issues and component updates in the future. The reports' frequency with this information should be customizable to be adapted to the duration of a sprint. The visualization seems practical to him. On the last level, he would like to mitigate the obligations, e.g., by using a checkbox directly. Then the status should be recalculated. Another aspect of the future are suggestions for components.

- **Interview 8**

  For the last ten years, the interviewee acted as an adviser and trainer for Software Architects or System Architects. Before, he primarily worked as a Software Architect.

  He considers himself an OSS expert, as he is both performing active OSS development for 25 years on the one hand and massively leveraging from OSS components in the industry on the other hand for 20 years. He states that as an Architect, he is always responsible for the compliance checks. However, he usually does it formally, except when asked by the customer for a detailed report.

  Nevertheless, he always checks transitively in-depth the esoteric or newer components. Still, he glances over those components he already checked once in the past, even if it was a different version. He states that it is not perfect, but more investigation is usually not within the time budgets.

He usually knows the individual licenses "good enough", but forget lots of details because of the total amount of licenses in practice. Therefore, an overview would indeed be useful for him to recap the essential points of a license. Compared to his old process, the new approach is more strict and precise and also more formal.

He considers the preciseness as an advantage, but the formal nature and the fact that one still has to enter lots of details are not optimal and states that it can be considered a disadvantage as it might be a nasty burden in practice. Another advantage he states is that the license analysis is no longer in the head of an architect, but permanently stored and available in a structured way. He states that this especially allows reports to be generated more quickly, and he hopes the tool can simplify the overall process.

The visualization he considers as looking cool and very useful. He states that it allows one to easily drill down to the origins of the license compliance trouble in the usual deeply nested component scenarios. An improvement he suggests is the comparison of a new and an old state. Currently, the visualization is only for a particular scenario. If the scenario changes, because component versions changed, he states that it would be cool to see just the actual difference against an older version to drill down to only these differences instead of drill down on all red cells from scratch.

Another improvement he suggests is the integration into the build process of an application. He states that this way, the architect would not have to fill information into a portal, but could keep the master information inside the source tree.

Even if he desires full automation through the build process, he considers it not as feasible. About a decade ago, he already tried such an approach and failed because it is not trivial to map from a component version to the precise license. He said it might be easier as SPDX became more feasible, but the guesses one might fail due to imprecise automated determination of information.

- **Interview 9**

  The interviewer is mainly responsible for information security, but he also considers to check license compliance within his process. There is no previous process. His responsibility is to establish a process to check the components, mainly for vulnerabilities and license compliance.

  For the license overview, he considers an indicator for licenses in the form of a time bar or a tachometer. A traffic light, he states, as tricky as it only has absolute values. Furthermore, he desires information about risk acceptance and statement on the risk level based on statistics when using the component when not complying with all the license terms. Further license information is not required since he would trust the algorithm. After the evaluation, he would desire a step by step description who guides him to comply with the license terms.

He also noted the help request as positive because it allows you to automatically request help within the tool without searching for a suitable contact person. He considers the user-friendly and understandable presentation of the use types as very important because if the results are not correctly understood, the results could be incorrect. He also considers the input and maintenance of the information to be time-consuming.

The reports are considered as very useful. The interviewee states that it can also be used by the legal department or the compliance team as a basis for making decisions about what risks they are willing to take.

He likes the idea of visualization. However, he misses context information when displaying the results, e.g., the license information with the specified use types for a red component. Furthermore, he wants a complete overview of the trouble-causing components without clicking through the whole tree.

Besides, he states that automation would be essential. Therefore he suggests that a warning should be displayed when installing a component. Suppose the command "npm install component" is entered, and the installed component stays under a problematic license. In that case, a warning should be displayed, which indicates that the license is causing problems, with the exception that the license is, e.g., used internally, which usually eliminates the copyleft effect. The user should then confirm by entering "yes" when the component should still be installed. This workflow would ensure that the component would not have to be removed again, but would be checked right at the beginning of the integration.

- **Conclusion**

  My conclusion is that we have created a reasonable basis for a tool that supports the architects in the license compliance process. I share Interviewer 5's statement that the selection of components must happen before integration. However, the process should be automated for agile teams where something is constantly changing. I also share the Interviewer's statement that a defined process is essential, including the verification of licenses using this tool. The component information could also be read in automatically in a further step, significantly reducing the manual effort. However, one would have to make sure that the data is correct since the architects rely on the data's correctness. The current solution solves this through the approval process, which means that experts confirm the information. I also consider the notifications on security issues and component updates as very useful. The more added value the system provides for the architects, the more motivated they are to follow the process. Furthermore, if the information is already stored in a central system, the benefits should be used. Besides, providing such information to the customer and reports on the components used and

the associated licenses makes the company look competent. There is also a potential for improvement for the treemap. I like the idea for different views from Interviewer 6, so everyone could choose the view that gives them more advantages. Also, I like the idea of integrating mitigation into the treemap, as Interviewer 7 suggested. So all information would be contained in the treemap.

7 Conclusion

7.1 Lessons learned

Architects are overwhelmed with the task and are looking for help. One indication of this was the request for support for a customer project. The customer was a large automotive manufacturer who requested information regarding the OSS components in their product.

Within a bachelor thesis, it is not possible to plan and implement the system simultaneously. Although it looks like a small application, it is still a whole business information system challenging to plan. Through many iterations, the understanding of the system grows and leads to new ideas and improvements.

Designing a data model to represent the real world is challenging. It requires many iterations and is, therefore, continuously improved. Moreover, the limits of modeling are quickly reached. An example demonstrates the formula, which is attached to the entity License Reference (Figure 4.5) by a comment.

7.2 Limitations

The solution of the bachelor thesis is limited by aspects such as time and limited access to information. For example, it is not possible to include and interview an unlimited number of organizations and persons within the given time, limiting the amount of information.

The collection of requirements through interviews is also subjective, as is the evaluation. To obtain meaningful information from interviews, this tool's users should be asked regularly after the development is completed. As the tool is not used daily, it is recommended to interview different people in intervals of 2-6 months to evaluate the experiences and identify the problems in different projects. Additionally, the implementation was outside the bachelor thesis scope due to the time limit, making the evaluation difficult. Furthermore, it was impossible to realize all requirements, e.g., automation, within the bachelor thesis.

7.3 Future work

For future works one could consider to focus more on automatization, since the basis is already proofed. Therefore, a scanner can be implemented, which disassembles the product and determines the structure. This should automatically evaluate a component. The architect

should not have to make any manual specifications regarding all components that the product must contain. Also, the tool could generate a graphical report in the form of an architecture diagram on request.

The integration of Common Vulnerabilities and Exposures (CVE) security numbers can inform users of security issues. CVE is a common database that provides information about security issues. The system should automatically determine which product is affected by a security issue and inform the corresponding product owner and/or architects. However, to get information about a CVE report, machine learning must be applied. Alternatively, the National Vulnerability Database (NVD) can be used to provide CVE information in a structured form.

Integration of third-party tools and external databases can be integrated to keep the information up to date and reduce manual effort. For example, if a user in the company looks up a related topic in a search engine such as Google, the information of the tool could be used to deliver suitable results.

The functionality to recommend components based on the number and frequency of security issues, the number of GitHub likes and ratings, the number of releases and over what period they were released, the number of dependencies a component contains through the integration of other components, and the number of people using the component can additionally be developed.

A metric for the strictness of the licenses could be computed to enable users to estimate the licenses better. Therefore, the values assigned in the solution from the UseTypeObligation combination can be used. The following graphs show examples based on the MIT (Figure 7.1), Apache-2.0 (Figure 7.2) and AGPL-3.0 license (Figure 7.3). The formula f(x) = 7 - x was applied to each cell of the adjacency matrix. On the left the original matrix is shown and on the right the matrix after applying the formula. However, this method needs to be checked for additional licenses to verify the validity and validity of the general public.

	format: source	format: compiled	dependency: optional	dependency: mandatory	delivery: internal	delivery: distributed	usage: local	usage: remote-call	communication: process	communication: system	bundling: standalone	bundling: embedded	artifact: pristine	artifact: modified
NO-LIABILITY	5	5	5	5	6	5	5	5	5	5	5	6	5	6
KEEP-COPYRIGHT	5	5	5	5	5	5	5	5	5	5	5	5	5	5
PROVIDE-LICENSE	5	5	5	5	5	5	5	5	5	5	5	5	5	5
PROVIDE-SOURCE														
ADV-CLAUSE														
RENAME														
NO-RELICENSE														
CTX-NON-MIL														
CTX-NON-COM														
COPYLEFT-STRONG														
COPYLEFT-WEAK														
NON-OSS-DEF														
OTHER														

	format: source	format: compiled	dependency: optional	dependency: mandatory	delivery: internal	delivery: distributed	usage: local	usage: remote-call	communication: process	communication: system	bundling: standalone	bundling: embedded	artifact: pristine	artifact: modified
NO-LIABILITY	2	2	2	2	1	2	2	2	2	2	2	1	2	1
KEEP-COPYRIGHT	2	2	2	2	2	2	2	2	2	2	2	2	2	2
PROVIDE-LICENSE	2	2	2	2	2	2	2	2	2	2	2	2	2	2
PROVIDE-SOURCE														
ADV-CLAUSE														
RENAME														
NO-RELICENSE														
CTX-NON-MIL														
CTX-NON-COM														
COPYLEFT-STRONG														
COPYLEFT-WEAK														
NON-OSS-DEF														
OTHER														81

Figure 7.1: MIT Metric Calculation

	format: source	format: compiled	dependency: optional	dependency: mandatory	delivery: internal	delivery: distributed	usage: local	usage: remote-call	communication: process	communication: system	bundling: standalone	bundling: embedded	artifact: pristine	artifact: modified
NO-LIABILITY	5	5	5	5	6	5	5	5	5	5	5	6	5	6
KEEP-COPYRIGHT	5	5	5	5		5	5	5	5	5	5	5	5	5
PROVIDE-LICENSE	5	5	5	5		5	5	5	5	5	5	5	5	5
PROVIDE-SOURCE														
ADV-CLAUSE	5	3	3	3		3	3	3	3	3	4	3	3	3
RENAME	5	5	5	5		5	5	5	5	5	5	5	5	4
NO-RELICENSE	4	4	4	4		4	4	4	4	4	4	4	4	
CTX-NON-MIL														
CTX-NON-COM														
COPYLEFT-STRONG														
COPYLEFT-WEAK														
NON-OSS-DEF														
OTHER														

	format: source	format: compiled	dependency: optional	dependency: mandatory	delivery: internal	delivery: distributed	usage: local	usage: remote-call	communication: process	communication: system	bundling: standalone	bundling: embedded	artifact: pristine	artifact: modified
NO-LIABILITY	2	2	2	2	1	2	2	2	2	2	2	1	2	1
KEEP-COPYRIGHT	2	2	2	2		2	2	2	2	2	2	2	2	2
PROVIDE-LICENSE	2	2	2	2		2	2	2	2	2	2	2	2	2
PROVIDE-SOURCE														
ADV-CLAUSE	2	4	4	4		4	4	4	4	4	3	4	4	4
RENAME	2	2	2	2		2	2	2	2	2	2	2	2	3
NO-RELICENSE	3	3	3	3		3	3	3	3	3	3	3	3	
CTX-NON-MIL														
CTX-NON-COM														
COPYLEFT-STRONG														
COPYLEFT-WEAK														
NON-OSS-DEF														
OTHER														189

Figure 7.2: Apache-2.0 Metric Calculation

	format: source	format: compiled	dependency: optional	dependency: mandatory	delivery: internal	delivery: distributed	usage: local	usage: remote-call	communication: process	communication: system	bundling: standalone	bundling: embedded	artifact: pristine	artifact: modified
NO-LIABILITY	5	5	5	5	6	5	5	5	5	5	5	6	5	6
KEEP-COPYRIGHT	5	5	5	5	5	5	5	5	5	5	5	5	5	5
PROVIDE-LICENSE	5	5	5	5		5	5	5	5	5	5	5	5	5
PROVIDE-SOURCE		4	4	4		4	4	4	4	4	4	4	5	4
ADV-CLAUSE														
RENAME														
NO-RELICENSE	4	4	4	4		4	4	4	4	4	4	4	4	3
CTX-NON-MIL														
CTX-NON-COM														
COPYLEFT-STRONG	1	1	1	1		0	1	1	1	1	1	0	1	0
COPYLEFT-WEAK														
NON-OSS-DEF														
OTHER	3	3	3	3	4	2	3	3	3	3	3	3	3	3

	format: source	format: compiled	dependency: optional	dependency: mandatory	delivery: internal	delivery: distributed	usage: local	usage: remote-call	communication: process	communication: system	bundling: standalone	bundling: embedded	artifact: pristine	artifact: modified	
NO-LIABILITY	2	2	2	2	1	2	2	2	2	2	2	1	2	1	
KEEP-COPYRIGHT	2	2	2	2	2	2	2	2	2	2	2	2	2	2	
PROVIDE-LICENSE	2	2	2	2		2	2	2	2	2	2	2	2	2	
PROVIDE-SOURCE		3	3	3		3	3	3	3	3	3	3	2	3	
ADV-CLAUSE															
RENAME															
NO-RELICENSE	3	3	3	3		3	3	3	3	3	3	3	3	4	
CTX-NON-MIL															
CTX-NON-COM															
COPYLEFT-STRONG	6	6	6	6		7	6	6	6	6	6	7	6	7	
COPYLEFT-WEAK															
NON-OSS-DEF															
OTHER	4	4	4	4	3	5	4	4	4	4	4	4	4	4	291

Figure 7.3: AGPL-3.0 Metric Calculation

8 Appendix

In the back of this bachelor thesis a CD is attached with the following documents:

Filename	Description
license-modeling.xlsx	License document of the 18 mentioned licenses in subsection 4.3.2
licenses-to-json-format.js	Script to convert license analysis into json format.
license-declarations.json	License Declarations from License Analysis in json format.
Defcon.js	JavaScript File with the Algorithm as described in section 5.2
bootstrap-business-test.json	Component Usage as described in section 5.3
spring-boot.json	Component Usage as described in section 5.3
test.js	Test JavaScript file to run the algorithm
output.json	Generated output after running the test.js file.
datamodel.pdf	Datamodel of the application Figure 4.5.
bpmn.pdf	Business Process Model of the application process Figure 4.27.

Acronyms

AGPL-3.0 GNU Affero General Public License, Version 3. 37, 44, 90

Apache-1.1 Apache Software License 1.1. 37

Apache-2.0 Apache Software License 2.0. 37, 71–74, 85, 90

BSD-2-Clause The 2-Clause BSD License. 37, 40, 41, 46

BSD-3-Clause The 3-Clause BSD License. 37, 72

BSD-4-Clause The 4-Clause BSD License. 37, 47, 48

CDDL-1.0 COMMON DEVELOPMENT AND DISTRIBUTION LICENSE Version 1.0. 37

CPL-1.0 Common Public License, version 1.0. 37

EPL-1.0 Eclipse Public License - v 1.0. 37, 72, 74, 75

GPL-2.0-only GNU General Public License, Version 2. 16, 37, 42, 43, 50, 51

GPL-3.0-only GNU General Public License, Version 3. 37, 43, 44, 50, 51

ICU ICU License. 37

LGPL-2.0-only GNU Lesser General Public License, Version 2. 37, 42, 43, 49, 50

LGPL-3.0-only GNU Lesser General Public License, Version 3. 37, 43–45

MIT The MIT License. 39, 40, 71, 80, 85, 90

MPL-2.0 MOZILLA PUBLIC LICENSE VERSION 2.0. 37

OSS Open Source Software. iii, viii, 1, 2, 6, 7, 10–14, 16, 17, 19, 20, 35, 49, 65, 71, 81, 85, 89

OTN Oracle Technology Network License Agreement. 37, 48, 49

SDN SAP Developers Network MaxDB License Agreement Version 1. 37, 49

Bibliography

[1] D. R. S. Engelschall. “ARCHITEKTUR VS. LIZENZRECHT: LIZENZKONFORME VERBAUUNG VON OPEN-SOURCE-SOFTWARE”. In: (2012).

[2] A. R. Hevner, S. T. March, J. Park, and S. Ram. “DESIGN SCIENCE IN INFORMATION SYSTEMS RESEARCH”. In: (2004).

[3] D. R. S. Engelschall. “Archicture Fundamentals”. In: (2009-2020). URL: https://bit.ly/2YJBWGQ (visited on 09/25/2020).

[4] N. Hansen. “Wirtschaftsinformatik 1, UTB Verlag, 10. Auflage”. In: (2009).

[5] P. D. F. Matthes. “Lecure 6 of Software Engineering in Business Applications: "Technische Grundlagen betrieblicher IS"”. In: (2019).

[6] OSI/IEC. “Systems and software engineering — Vocabulary”. In: (2017). URL: https://bit.ly/3i5cNgX.

[7] A. Silberschatz, P. B. Galvin, and G. Gagne. “Operating System Concepts, Ninth edition”. In: (2012).

[8] R. Stallman. “Free Software Definition”. In: (1996-2019). URL: https://bit.ly/3iPwLwP (visited on 08/05/2020).

[9] bitkom. “Am Anfang war alle Software frei”. In: (2016). URL: https://bit.ly/3gVjlOf (visited on 09/03/2020).

[10] R. Stallman. “Licenses”. In: (2014-2020). URL: https://bit.ly/3fsTb4C (visited on 08/05/2020).

[11] O. Sun Microsystems. “Common Development and Distribution License 1.0”. In: (N/A). URL: https://bit.ly/2E8hyrD (visited on 08/05/2020).

[12] Mozilla. “Mozilla Public License Version 2.0”. In: (2012). URL: https://mzl.la/2Y85bml (visited on 08/05/2020).

[13] Eclipse. “Eclipse Public License - v2.0”. In: (N/A). URL: https://bit.ly/2E4wOWv (visited on 08/05/2020).

[14] IBM. “Common Public License, version 1.0”. In: (N/A). URL: https://bit.ly/3ib5hkG (visited on 08/05/2020).

[15] R. Stallman. “GNU General Public License, Version 3”. In: (2007). URL: https://bit.ly/39WGYUQ (visited on 08/05/2020).

[16] R. Stallman. “GNU Affero General Public License, Version 3”. In: (2007). URL: https://bit.ly/2PrJj0M (visited on 08/05/2020).

[17] R. Stallman. "GNU Lesser General Public License, Version 3". In: (2007). URL: https://bit.ly/30wVVKl (visited on 08/05/2020).

[18] msg systems ag. "msg". In: (2020). URL: https://bit.ly/2YP5DWZ (visited on 08/30/2020).

[19] Snopsys. "Black Duck". In: (2019). URL: https://bit.ly/3hp9cdL (visited on 08/23/2020).

[20] WhitSource. "WhiteSource". In: (2020). URL: https://bit.ly/3go1EXi (visited on 08/23/2020).

[21] FOSSA. "FOSSA". In: (2020). URL: https://bit.ly/31nO7Ln (visited on 08/23/2020).

[22] SPDX. "SPDX License List". In: (2018). URL: https://bit.ly/31lmjXN (visited on 08/23/2020).

[23] tl;dr Legal. "tl;dr Legal". In: (2012-2017). URL: https://bit.ly/32fBBwt (visited on 08/23/2020).

[24] PERFORCE. "OpenLogic". In: (2020). URL: https://bit.ly/3aMogzN (visited on 08/23/2020).

[25] Eclipse. "Eclipse SW 360". In: (2018). URL: https://bit.ly/31mvXK0 (visited on 08/23/2020).

[26] FOSSology. "FOSSology". In: (2017). URL: https://bit.ly/2YsowyW (visited on 08/23/2020).

[27] H. Zhang, B. Shi, and L. Zhang. "Automatic Checking of License Compliance". In: *2010 IEEE International Conference on Software Maintenance* (2010), pp. 1–3. DOI: 10.1109/icsm.2010.5609557.

[28] D. M. German and M. D. Penta. "A Method for Open Source License Compliance of Java Applications". In: *IEEE COMPUTER SOCIETY* (2012).

[29] H. Y. Yun, Y. J. Joe, and D. M. Shin. "Method of License Compliance of Open Source Software Governance". In: (2017).

[30] D. O. Block. "Assessing Open Source Software Usage in the Development of Grid Control and Measurement Device Software". In: *2012 International Conference on Smart Grid Technology, Economics and Policies (SG-TEP)* (2012), pp. 1–3. DOI: 10.1109/sg-tep.2012.6739584.

[31] "The JSON License". In: (2002). URL: https://bit.ly/2EcDJww (visited on 08/16/2020).

[32] "Prototype". In: (2005-2007). URL: https://bit.ly/2E9vilE (visited on 08/16/2020).

[33] T. Fuchs. "Scriptaculous". In: (2005-2008). URL: https://bit.ly/31VZTeR (visited on 08/16/2020).

[34] "SLF4J". In: (2004-2017). URL: https://bit.ly/3hdXPFi (visited on 08/16/2020).

[35] "W3C SOFTWARE NOTICE AND LICENSE". In: (2015). URL: https://bit.ly/3h2emMt (visited on 08/16/2020).

[36] "ASM". In: (2000-2011). URL: https://bit.ly/340d7tM (visited on 08/16/2020).

[37] T. H. D. Group. "HSQLDB". In: (2001-2020). URL: https://bit.ly/3h5h2Ji (visited on 08/16/2020).

[38] S. Souza. "JAMon License Agreement". In: (2002). URL: https://bit.ly/2PZQ9uB (visited on 08/16/2020).

[39] T. P. G. D. Group. "PostgreSQL". In: (1996-2020,). URL: https://bit.ly/2Y4qGo6 (visited on 08/16/2020).

[40] M. Ltd. "dom4j Copyright and License Agreement". In: (2001-2016). URL: https://bit.ly/3g2FGZG (visited on 08/16/2020).

[41] T. V. S. Society. "FreeMarker". In: (2003). URL: https://bit.ly/2Y8IQFr (visited on 08/16/2020).

[42] G. Robert A. van Engelen. "gSOAP Public License". In: (2001-2009). URL: https://fla.st/2PWphvr (visited on 08/16/2020).

[43] S. AG. "SAP DEVELOPERS NETWORK –MaxDB LICENSE AGREEMENT". In: (N/A). URL: https://bit.ly/3iKpqhL (visited on 08/16/2020).

[44] M. I. of Technology. "The MIT License". In: (N/A). URL: https://bit.ly/3fDfQM1 (visited on 08/05/2020).

[45] R. of the University of California. "The 2-Clause BSD License". In: (1999). URL: https://bit.ly/33ucLuY (visited on 08/05/2020).

[46] R. of the University of California. "The 3-Clause BSD License". In: (1999). URL: https://bit.ly/30riVu6 (visited on 08/05/2020).

[47] R. of the University of California. "The 4-Clause BSD License". In: (1999). URL: https://bit.ly/2DplvYT (visited on 08/05/2020).

[48] "ICU License". In: (1995-2009). URL: https://bit.ly/2DxYAuf (visited on 08/05/2020).

[49] A. S. Foundation. "Apache Software License 1.1". In: (2001). URL: https://bit.ly/2ESVUrI (visited on 08/05/2020).

[50] A. S. Foundation. "APACHE LICENSE, VERSION 2.0". In: (2004). URL: https://bit.ly/2PnbLAG (visited on 08/05/2020).

[51] R. Stallman. "GNU Lesser General Public License, Version 2". In: (1991). URL: https://bit.ly/2QZYwXz (visited on 08/05/2020).

[52] R. Stallman. "GNU General Public License, Version 2". In: (1989-1991). URL: https://bit.ly/33ufXGY (visited on 08/05/2020).

[53] Oracle. "Oracle Technology Network License Agreement". In: (2014). URL: https://bit.ly/2Ptf2i1 (visited on 08/05/2020).

[54] SAP. "SAP DEVELOPERS NETWORK –MaxDB LICENSE AGREEMENT Version 1". In: (2007). URL: https://bit.ly/31kE2xs (visited on 08/05/2020).

[55] R. Stallman. "Frequently Asked Questions about the GNU Licenses". In: (2014-2020). URL: https://bit.ly/3iT56uW (visited on 08/18/2020).

[56] P. D. D. Riehle. “Open Source License Compliance and Work-for-Hire”. In: (2020). URL: https://bit.ly/3iYnItz (visited on 08/18/2020).

[57] P. D. D. Riehle. “Open Source License Compliance and Work-for-Hire”. In: (2020). URL: https://bit.ly/3h9wxjn (visited on 08/18/2020).

[58] T. G. Foundation. “GraphQL”. In: (2020). URL: https://bit.ly/3jxbgAX (visited on 08/30/2020).

[59] Pivotal. “Spring Boot » 2.3.3.RELEASE”. In: (2006-2020). URL: https://bit.ly/3iQv6XD (visited on 08/18/2020).

[60] F. Daugan. “Bootstrap Business Test » 2.8.3”. In: (2006-2020). URL: https://bit.ly/3ha4QXP (visited on 08/18/2020).