An Organizational-Technical Concept to Deal with Open Source Software License Terms

Sergius DYCK, Daniel HAFERKORN, and Jennifer SANDER
Fraunhofer IOSB, Karlsruhe, Germany

ABSTRACT

Open source software (OSS) released under various license terms is widely used as third party libraries in today's software projects. To ensure open source compliance within an organization, a strategic approach to OSS management is needed. As basis for such an approach, we introduce an organizational-technical concept for dealing with the various OSS licenses by using procedural instructions and build automation software. The concept includes the careful consideration of OSS license conditions. The results obtained from this consideration and additionally necessary commitments are documented in a so-called "license playbook". We introduce procedure instructions enabling a consistent approach for software development using OSS libraries. The procedure instructions are described in a way such that they can be implemented for example for Java projects using the popular build automation tool Apache Maven and the software repository tool Nexus. We give guidance on how to realize such an implementation on basis of automation tools in practice.

Keywords: open source software, open source compliance, organizational-technical concept, procedure instructions, build automation software, software engineering.

1. INTRODUCTION

Today's software projects are often large and complex. Usually, they make use of various third party libraries to reduce both considerable engineering effort and development time. Depending on the project, the software often includes many third party libraries released under different OSS licenses. More than 65% of companies surveyed in [1] use OSS in application development.

OSS possesses some essential characteristics [2]. Any party has the right to use the software for its specific purpose. It is allowed to modify the software, to generate derived works, and to transfer the software to third parties. The source code of the software must be available according to the conditions of many OSS licenses. The respective OSS license applies to parties to which the OSS is distributed. The OSS license grants the right to use the OSS and, in return, a party using the OSS must comply with the obligations according to the OSS license.

Many different OSS licenses are used for OSS projects. The spectrum ranges from very popular OSS licenses, used in many practical application fields and coordinated by a wider community, to specific OSS licenses, used only by single OSS projects that may be rather specialized. OSS licenses grant different concrete rights and impose different obligations to the using party [3].

Each OSS license imposes at least minimal obligations which have to be adhered to by the parties using the corresponding OSS. Depending on the implications of these obligations, usually, one considers an OSS license as being more or less restrictive. A common way to differentiate OSS licenses in regard to their restrictiveness is for example to distinguish between strong-copyleft licenses, weak-copyleft licenses and non-copyleft licenses [4].

If OSS is used within a certain software project of an organization, it must be ensured that all obligations to be adhered to according to the used OSS license are satisfied precisely. Any violation of these obligations constitutes a considerable risk for the organization. For example, such a violation may require cost-intensive changes of software projects or it may cause damage to the reputation of the organization [5].

In order to enable individual developers to make decisions with regard to the concrete use of software under specific OSS licenses, an organization may define its own OSS policy. Different skills, areas of knowledge, and responsibilities of software developers, executives, and legal counsels must be brought together and coordinated to define such a policy and to enable individual decisions in a well-founded and efficient way. Such a policy can serve as a basis for enabling specific case-to-case analyses and decisions.

To address open source compliance from a procedural point of view, responsibilities must be assigned to personnel having the appropriate positions for making corresponding decisions, being suitably skilled and possessing the respective knowledge (to be achieved for example by specific training activities). Thereby, one must be aware that a significant difficulty with regard to open source compliance is that the detailed analysis of the obligations imposed by OSS licenses addresses - at least for the concrete case - two domains, the technical domain and the legal domain. Usually, a legal counsel may provide practical advice with the aim to enable software developers to make individual decisions in their daily business. However, as the concrete consequences arising from the use of a certain OSS usually depend on the concrete manner in which the software is used within a software project, this advice cannot replace the technical expertise of the software developers. For example, in case of OSS licensed under a weak-copyleft license, only some kinds of modifications of the corresponding OSS have to be licensed under the original license when the software project using this library is distributed. The question whether the copyleft clause applies in the concrete case and which other parts of the corresponding software project are affected by this kind of "viral" effect cannot be answered without specific technical knowledge. In this case the precise form of the modifications as well as the specific use and integration of the OSS within the software project are of relevance. These technical details must be assessed by the software development team.

To ensure open source compliance, OSS license compatibility checks and the verification of fulfillment of all license conditions are essential, at least before releasing a software project. To ensure that all used OSS and the corresponding licenses as well as all copyright information are captured, and that the source code of the OSS is administered properly is of crucial importance.

In this context it is also worth mentioning that some OSS is released under several different OSS licenses as alternatives, from which one has to be selected. This makes open source compliance additionally difficult.

In principle, the used OSS and the accompanying OSS licenses could be handled manually within an organization. However, such an approach is error-prone, time-consuming, and requires a profound knowledge of the employed OSS licenses including an agreed general legal interpretation and the consideration of cross-linkages between them. Tools and automation are critical to ensure efficiency of the open source compliance process.

To address the described challenges, it is necessary to have a strategic approach to OSS management - at least on the level of an organizational unit. As a fundamental basis to that, we worked out an organizational-technical concept for ensuring open source compliance.

According to our investigation with regard to freely available tools to ensure open source compliance, these tools target mostly audit aspects and deliver reports or other kinds of analysis results that need to be manually processed. Although many of these tools can be included in a build automation setup, they do not ensure open source compliance by themselves – instead they merely detect possible open source compliance issues. In contrast, our approach aims for a solution that is firmly integrated in the software development and also ensures open source compliance in every phase of the software development lifecycle (SDL).

The concept we developed includes the careful consideration of all relevant OSS license conditions. These need to be analyzed and evaluated for their legal rights and obligations – where necessary also supported by legal experts. For ambiguous or unclear license conditions, decisions regarding how these cases are to be approached need to be made and documented. The documentation has to state precisely how the license conditions are to be interpreted and what rights and obligations they are deemed to represent. The results obtained and further conclusions should be documented in a so-called "license playbook" [6]. Such license playbooks are an easy-to-read and digest summary of OSS licenses intended for software developers to support them in SDL [7]. Our concept furthermore includes procedure instructions to enable a consistent approach when using OSS in software projects. The procedure instructions take into account the necessary steps for two use cases: introducing new libraries to a software project already adhering to this concept as well as upgrading existing so-called legacy projects, which are not yet rooted in this concept.

From the technical point of view, the procedure instructions are described in a way such that they can be implemented using the popular build automation tool Maven and the software repository tool Nexus. In addition, our concept intends for the license texts to be managed together with the respective OSS libraries and their source code in a software repository. This allows publishing the necessary license terms and source code archives together with the software deployment of a software project by the means of its Maven configuration.

The remainder of this publication is organized as follows. Sec. 2 addresses essential aspects of related work and already existing approaches for open source compliance from both the organizational and the technical point of view. In Sec. 3, we summarize the main objectives of our approach and discuss its organizational and technical aspects. In Sec. 4, the essentials of the procedural instructions being part of our approach are presented. In Sec. 5, we describe how such an implementation can be realized in practice using the OSS tools Maven and Nexus Repository. Finally, in Sec. 6, we give a conclusion and indicate topics for future work.

2. RELATED WORK AND SIMILAR APPROACHES

In order to handle OSS and the corresponding legal issues properly, an open source compliance management has to be established on an organizational level. [8] gives advice how this could be implemented, especially also with regard to establishing appropriate open source compliance procedures.

As part of the described process, an open source compliance manager should be appointed and an OSS review board should be set up. The open source compliance manager must have a solid understanding of all aspects of open source compliance as well as technical knowledge, as he acts as contact person between development staff and legal counsel. The OSS review board, directed by the open source compliance manager, would be typically responsible for coordinating the use of OSS in projects and products as well as for OSS reviews and audits prior to deployments/releases. An OSS review should consist of source code audits and dependency linkage audits to determine the OSS components being used in the software project that is to be released. The result of the review should be a report that lists all OSS components along with respective details and metadata such as the used version and the corresponding OSS license.

A manual review would be complex and time-consuming. Furthermore, a review should be considered a recurring event. Therefore, the establishment of an automated system for OSS audits is recommended by [8]. To this aim, several OSS organizations/initiatives as well as companies have released various analysis and reporting tools. We reviewed the ones that are relevant according to [9] in detail. Our emphasis lies on using OSS tools for achieving open source compliance. We thereby focused on OSS solutions and excluded commercially distributed tools consciously from the detailed analysis.

Many tools being available to support open source compliance are dealing with source code analysis and dependency linkage analysis to determine what OSS libraries have been used and which licenses have to be acknowledged and complied to. For example, the source code scanning tool FOSSology mentioned in [9] performs license, copyright, and export control scans on the source code and creates reports in the common Software Package Data Exchange (SPDX) format. These reports can be used to generate formatted license reports. The tool can also be integrated into software development and deployment lifecycles.

The Open Source License Checker mentioned in [9] works similar to FOSSology by trying to match source code files provided by the user to license texts from an internal database. However, it is not anymore in active development.

Another common open source compliance tool mentioned in [9] is the Binary Analysis Tool. It focuses on analyzing binary software packages such as Executable and Linkable Format (ELF) files, Android packages and Java classes and on detecting possible compliance issues in the binary code. This is done using pre-defined and customizable rules and an internal database with information extracted from the OSS code.

These three tools, FOSSology, Open Source License Checker and Binary Analysis Tool, are suited to support for example an open source compliance manager in the audit process prior to a release. But at this point, the OSS may have already been heavily integrated in a software project. Hence, in the worst case, the audit could reveal the necessity for cost-intensive changes to the software project due to the need to replace the OSS library, for example with another one than can be used without concerns regarding open source compliance.

To avoid scenarios like this, [6] suggests to define and to implement an internal process which forces the software developers to consult with an open source compliance manager prior to using an OSS component in a software project. However, in daily business, this may not always be a practicable solution, as it may for example delay the work to be done by the software developers. Therefore, a process is needed which ensures open source compliance and which works closely in an automated way within the SDL – such that the software developers are not constrained when adding OSS to their software projects and are able to do so using a well-defined process. The next sections describe such a process that we developed for this purpose.

3. OUR APPROACH

Our approach aims for a solution that is firmly integrated in the SDL. The resulting organizational-technical concept was designed according to three main objectives:

- to support the developers to deal with OSS in their daily work;
- to ensure compliance with regard to the license terms;
- to enhance the deployment process – especially in such a way that relevant OSS licenses and the source code of the used OSS libraries are packaged automatically.

The concept consists of organizational aspects and technical aspects. The organizational aspects include procedure instructions for software developers when planning to use new OSS in a software project that need to be followed manually, supplemented by a license playbook. The technical aspects allow the use of build automation tools to support and verify open source compliance during the SDL.

For each OSS library that is newly introduced into a software project, the procedure instructions specify a series of steps to check various conditions for the integration of the new OSS concerning its license terms and ensuring that any problems related to license conditions are being solved prior to its use. They are also designed to ensure that all the necessary adjustments and extensions to the project configuration have been made and that the new OSS library has been integrated into the software repository in a well-defined form.

Afterwards, the deployment process in the SDL has to be adjusted so that the result of the procedural instructions can be used as a basis for ensuring open source compliance during the deployment process. In our case, these technical adjustments resulted in changes to our Java projects that use Maven as a build automation and build lifecycle tool.

There are several possible ways to use Maven as a support tool for open source compliance. As a first approach, we initially looked for existing build tools and Maven plugins to support open source compliance as part of the SDL and found various similar plugins, for instance the License Maven Plugin. These tools make use of the metadata that is available in Maven dependencies to create dependency lists and license reports. They also address the common problem of the metadata often not conforming to official naming conventions or outright missing metadata by employing mechanisms for providing temporary replacement values for the original metadata values. However, we found that this usually needs to be done on a per-project basis. In consequence, having several projects using the same dependencies that need to be managed is redundant and error-prone. We therefore looked for other solutions.

A more feasible approach takes into account the repository from which these dependencies come from and stores the additional information in a way that can be used in a normal deployment process without complex processing or transformation. We modified our procedural instructions accordingly in several iterations to make use of this approach.

The procedure instructions of our concept are described in detail in Sec. 4 and the technical background using build automation software is detailed in Sec. 5.

4. PROCEDURE INSTRUCTIONS

The organizational aspects of our concept are the procedure instructions. They are specifically designed for software developers who want to add a new OSS library to one of their software projects. The procedure instructions are formulated in a way that no in-depth knowledge of OSS licenses is required for their steps. The process of *adding a new OSS* is described in the following by using flow charts. Simple and complex steps are being distinguished. Two of the complex steps are displayed within their own diagrams and described in more detail.

Fig. 1 depicts the process that must be followed when adding new OSS to a project. The first step is to check whether the required OSS is already contained in the local software repository. If this is not the case, the step *add new library* must be performed. This is a complex step being illustrated in Fig. 2 and described in detail in the next paragraph. If the OSS is already contained in the software repository, it has to be checked whether a corresponding license classifier file has been uploaded yet. The license classifier file contains the license text and is named as follows: *<groupid>-<artifactid>-<version>-license.txt*. On the one hand, this file is used as a flag to document the successful integration of the OSS library into the software repository. On the other hand, it is used to automatically gather the license files during a deployment process. See [10] for more information about the classifier tag. If the license classifier already exists, one can proceed to the next sub-process concerned with *fulfilling project specifics*, e.g., specific requirements. The absence of the license classifier indicates that the deployment of the library is not in the well-defined state. A separate sub-process handles such legacy libraries. These libraries have been introduced into the repository previously to establishing our organizational-technical concept and thus are regarded as legacy libraries. If one of the sub-processes *add new library* or *handle legacy library* has to be aborted, i.e., cannot be processed successfully, the entire process of adding a new OSS is immediately terminated. The library cannot be used then. One such example is that an internal policy might forbid the use of strong-copyleft licenses. When the respective sub-process has been successful, the license classifier was uploaded to the repository as a result. Afterwards, it has to be ensured that the project specific requirements are being fulfilled. If this sub-process is also successful, the usage of the desired OSS library is allowed and the library can be uploaded to the repository, otherwise, this is not the case.

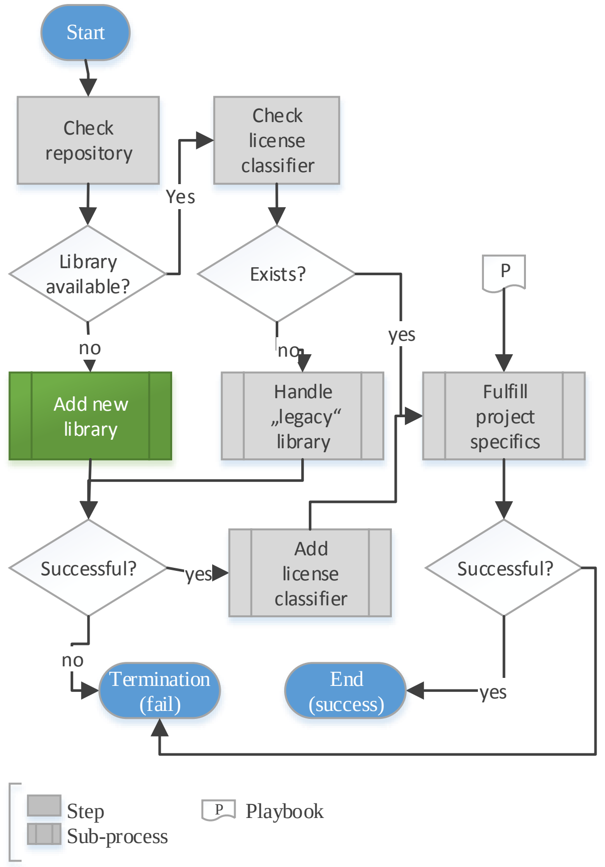

Figure 1 – Flow chart depicting the process “*Add new OSS*” defined in the procedure instructions.

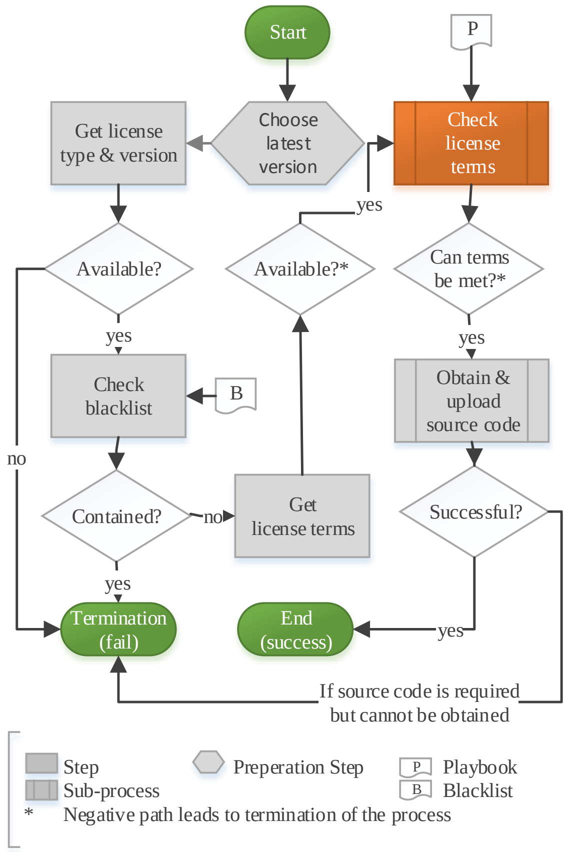

Figure 2 – Flow chart depicting the sub-process “*Add new library*” used in the process “*Add new OSS*”.

Fig. 2 shows a more detailed view on the sub-process used to *add a new library*. Before getting started, it shall be checked if a newer version of the OSS library is available. If possible, the latest version should be chosen. This is not possible, e.g., when a serious error is known to exist in the latest version or project-specific constraints require a particular, older version. As a next step, the type and version of the license under which the library was released have to be obtained. If this information is missing and cannot be discovered, it is not possible to check the license conditions. In this case, the process has failed and the desired library cannot be used. In case of successfully obtaining license type and version, it has to be checked if the license is contained in a blacklist. If this is the case, the process is terminated, too. If the license is not listed in the blacklist, the license terms have to obtained, e.g., as a text document. In most cases, these license terms are provided within the library distribution, in other cases, if the maintainer of the OSS library was remiss to provide a license document, an online search for the license terms can be necessary. If the license terms cannot be obtained, the process is again terminated, as it is not possible to check whether the library can be used or not. After successfully obtaining the license terms, these have to be checked in the next step. This sub-process will be described in detail in the next paragraph. If the checking is negative, the process fails. Otherwise, the sub-process of downloading the source code and uploading it to the software repository has to be executed. If the source code is required for delivery according to the license terms but cannot be obtained (e.g., from the Maven Central projects website, author's website), the process is again terminated as a fail.

As indicated in Fig. 3, two actors are directly involved in the sub-process *check license terms*. As a first step of this sub-process, the developer checks the license terms using a license playbook. License playbooks are an easy-to-read and digest summary of OSS licenses intended for software developers [6]. In case of licenses that are rather uncomplicated (like BSD or MIT license), the developer may decide by himself if the investigated license is permissive or not, and therefore if it can be used in a project or not. Afterwards, he informs the open source compliance manager about his decision. The open source compliance manager verifies the decision. In the case that he confirms the decision, the sub-process *check license terms* ends successfully. If making a decision by the developer on its own is not possible, he just informs the open source compliance manager who then checks the license terms by himself. For getting first hints, he may check the license terms using external sources like the *Open Source License Compendium* [11]. In addition, the license terms have to be checked against organizational guidelines, commitments and policies. If necessary, the open source compliance manager asks for support by the legal counsel before he makes his decision. If he decides that the use of the library is not allowed, the license type and version are added to the blacklist and the sub-process fails. If he decides that the use is allowed and if the license is new (i.e., firstly used), the corresponding license is added to the license playbook and the sub-process ends successfully.

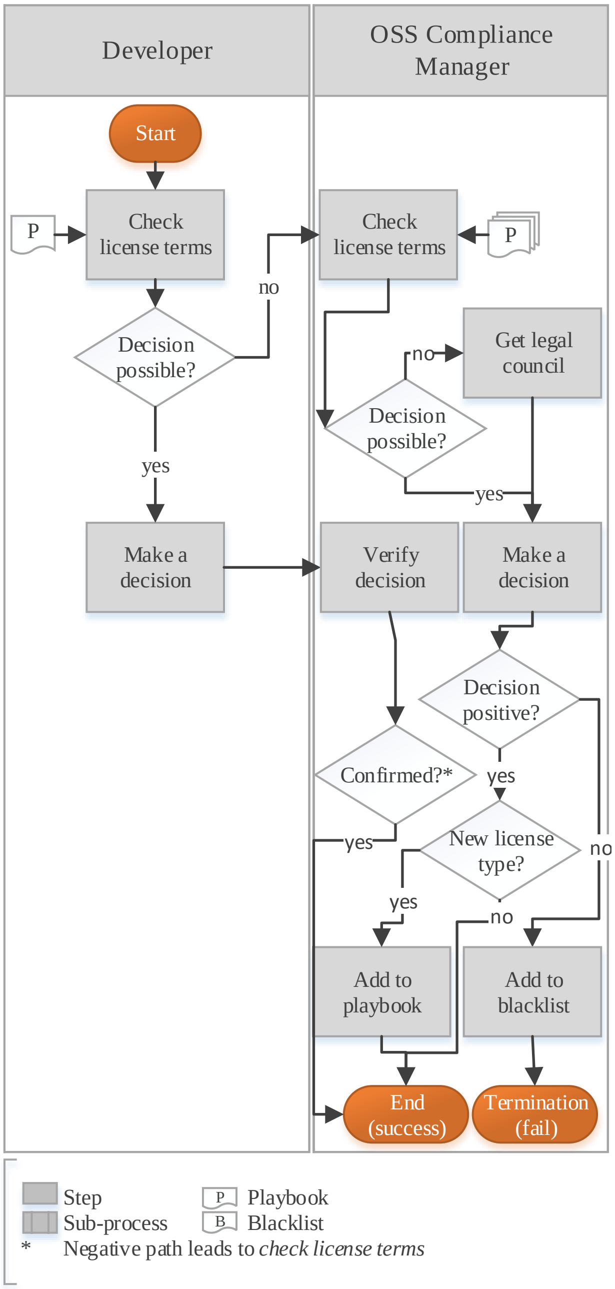

Figure 3 – Flow chart depicting the sub-process "*Check license terms*" used in the process "*Add new library*".

In addition to the process for adding new OSS libraries to a software project that has been described so far, a similar process has been defined for upgrading legacy projects.

These procedure instructions are a necessary foundation to automate the deployment. In the following section, it is detailed how an automated deployment can be realized.

5. USE OF BUILD AUTOMATION SOFTWARE

Our approach relies heavily on the SDL and the software development tools that we already use in practice. The build automation tool Maven is used in all our current projects. During a typical build lifecycle, it fetches project-specific OSS dependencies from a software repository. According to best practice, we have established a proxy repository for caching these external dependencies which is also used to manage internal dependencies and build artifacts, using the OSS Nexus Repository.

A typical Maven repository is not only used to manage OSS binaries, but also to store the source code of the OSS. Maven differentiates between the OSS artifacts themselves and related files by so-called classifiers, such as "javadoc" for documentation and "sources" for source code of the OSS. Additional classifiers for specific related file types may be used.

In practice, some project maintainers do not upload the source code of their OSS for undisclosed reasons. In order to make sure that all source code for a specific software release can be downloaded from the same source, a hosted repository has been added to the local Nexus installation, where source code not provided by the Maven central repository is uploaded and managed locally.

Similarly to the additional hosted repository for self-managed source code artifacts, another hosted repository has been added to serve as a storage and management facility for all OSS license files in plain text format, using the "licenses" classifier.

These two hosted repositories provide the storage for license and source code files obtained through the procedure instructions. What remains to be done in order to ensure open source compliance during the deployment process in relation to the distribution of the software together with the OSS source code and license terms, is the configuration of a Maven-based project to automatically manage these artifacts of OSS dependencies during the SDL. Maven already provides a configurable plugin to pull data from the repository, the Maven Dependency Plugin.

This plugin is primarily used to download dependencies themselves for release and deployment purposes. It can however also be configured to download related files of a dependency, based on the given classifier. The configuration is done via a XML file, the so-called Project Object Model (POM) file. The following Listing 1 gives an example of such a plugin configuration:

```
<plugin>
    <artifactId>maven-dependency-plugin</artifactId>
    <version>2.10</version>
    <executions>
        <execution>
            <id>copyLicenses</id>
            <phase>package</phase>
            <goals>
                <goal>copy-dependencies</goal>
            </goals>
            <configuration>
                <classifier>licenses</classifier>
                <type>txt</type>
                <outputDirectory>licenses</outputDirectory>
                <includeScope>runtime</includeScope>
                <prependGroupId>true</prependGroupId>
                <excludeGroupIds>
                        com.example
                </excludeGroupIds>
                <failOnMissingClassifierArtifact>
                        true
                </failOnMissingClassifierArtifact>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Listing 1: Example dependency plugin configuration.

The displayed example configuration with the id “copyLicenses” shows an execution of the “copy-dependencies” goal of the plugin during the “package” lifecycle phase of a build process. The configuration details define that all declared runtime dependency files with the classifier “licenses” and the file type (= file extension) “txt” are to be put in the licenses folder, using the naming convention that also contains the group identifier (= fully qualified name). If the build also contains internal dependencies for which the related files of the respective classifier should not be copied, they may be excluded. In this example, this has been done for all dependencies of the group “com.example”.

In case where a dependency file with the appropriate classifier cannot be found, the build process is set to fail, thus making the developer aware that there is an issue that needs to be resolved. If a continuous build environment with an appropriate alarm and notification setup has been put in place, the open source compliance manager can also be automatically informed in case of issues during the build process relating to license texts and source code of OSS dependencies.

By adding another execution definition and using respective parameters in the plugin configuration, the source code of all declared dependencies can also be downloaded as part of the automated build process.

6. CONCLUSION AND FURTHER WORK

In this contribution, an organizational-technical concept addressing OSS license terms was presented. Our aim was to define and to implement a solution for open source compliance that is firmly integrated in the SDL. Therefore, the created organizational-technological concept combines different means. Firstly, it defines processes that enable software developers to include OSS libraries properly into software projects while ensuring open source compliance under minimal involvement of specialized professionals such as an open source compliance manager or legal counsels. These processes are implemented as procedural instructions based on a process flow illustrated by flow charts and backed by a so-called license playbook, summarizing the most important aspects for handling the terms of different types of licenses. Secondly, for ensuring open source compliance on the technical side, the employment of existing build automation tools and software component management tools for this purpose is made possible by several technical extensions to the software deployment process.

Several key points are planned to be addressed with regard to future work:

- Practical tests conducted so far for single software developers have shown that additional refinements and a more detailed description of the steps in the procedure instructions may be necessary in order to make the processes easier to adhere to.
- On the technical side, means to generate OSS usage reports including details such as the OSS name, its version and the given license are also considered to be necessary in the future.
- Furthermore, the same metadata that is being used for the report generation could be employed to automatically generate acknowledgements and attributions for end-user documentations as well as possible splash or info screens for the compiled software project.
- In addition, it is planned to enhance the license playbook by a license compatibility matrix which will serve as a simple view to show incompatibilities between licenses (as described by [12]).
- Finally, the implementation of categorizations of OSS licenses in the license playbook, for example according to the scheme non-copyleft, weak-copyleft, strong-copyleft licenses, may be helpful.

7. REFERENCES

[1] **2016 Future of Open Source Survey Results**, Black Duck Software, 2016, http://de.slideshare.net/blackducksoftware/2016-future-of-open-source-survey-results/13 (date accessed: June 1, 2016)

[2] T. Jaeger, A. Metzger, **Open Source Software - Rechtliche Rahmenbedingungen der Freien Software**, Verlag C.H. Beck, 3. Edition, 2011

[3] A. M. St. Laurent, **Understanding Open Source and Free Software Licensing,** O'Reilly Media, 2004.

[4] C. Subramaniam, R. Sen, M. L. Nelson, **Determinants of open source software project success: A longitudinal study**, Decision Support Systems, Vol. 46, Issue 2, January 2009, pp. 576-585

[5] I. Haddad, **Open Source Compliance**, 2014, http://de.slideshare.net/SamsungOSG/guide-to-open-source-compliance/41 (date accessed: June 1, 2016)

[6] I. Haddad, **7 Steps to Strengthen Your Open Source Compliance**, Samsung Open Source Group Blog, 2015, https://blogs.s-osg.org/7-steps-to-strengthen-your-open-source-compliance/ (date accessed: May 20, 2016)

[7] M. Dolan, **5 Practical Ways for Legal Counsel to Advise Developers on Open Source**, Linux Blog, 2015, https://www.linux.com/blog/5-practical-ways-legal-counsel-advise-developers-open-source (date accessed: May 25, 2016)

[8] P. Koltun, **Free and Open Source Software Compliance: An Operational Perspective**, International Free and Open Source Software Law Review, Vol. 3, No. 1, 2011, pp. 95-101

[9] **Resources for Open Source Compliance**, Open source Initiative, https://opensource.org/node/539 (date accessed: May 31, 2016)

[10] J. Lalou, **Apache Maven Dependency Management**, Packt Publishing, 2013

[11] K. Reinckey, G. Sharpez, R. Dauster, **Open Source License Compendium**, Version 0.99.3, Deutsche Telekom AG and GIDO GmbH, 2014, http://opensource.telekom.net/-oscad/static/pdf/oslic-0.99.3.pdf (date accessed: May 31, 2016)

[12] T. Kreutzer, **Open Source Lizenzmanagement**, Vortrag bei den Kölner Tagen Informationsrecht, 2010, http://www.slideshare.net/littk/open-source-lizenzmanagement (date accessed: May 25, 2016)