Challenges of Tracking and Documenting Open Source Dependencies in Products: A Case Study

Andreas Bauer, Nikolay Harutyunyan, Dirk Riehle, and Georg-Daniel Schwarz

Friedrich-Alexander University Erlangen-Nuernberg, 91058 Erlangen, Germany
andi.bauer@fau.de, nikolay.harutyunyan@fau.de, dirk@riehle.org
georg.schwarz@fau.de

Abstract. Software vendors need to manage the dependencies of the open source components used in their products. Without this management, license compliance would be impossible, export restrictions could not be maintained, and security vulnerabilities would remain unknown to the vendor.
The management of these dependencies has grown in an ad-hoc fashion in most companies. As such, vendors find it hard to learn from each other and improve practices.
To address this problem, we performed exploratory single-case study research at one large established software vendor. We gathered and analyzed the key challenges of tracking and documenting open source dependencies in products. We wanted to understand whether these ad-hoc solutions could be based on a single unified conceptual model for managing dependencies.
Our study suggests that underlying the various point solutions that we found at this vendor lies a conceptual model that we tentatively call the product (architecture) model. In future cross-vendor work, we will investigate whether this conceptual model can be expanded to become a unifying model for all open source dependency management.

Keywords: Open Source Software · FLOSS · FOSS · Open Source Governance.

1 Introduction

The growth of free/libre, and open source software (FLOSS) leads the software industry to new opportunities but also challenges. FLOSS promise significant shortcuts by reusing existing software components in commercial products [15, 16, 13, 4, 1, 7]. However, to avoid legal and other risks of using FLOSS in commercial products, such as license noncompliance, software vendors need to manage their FLOSS dependencies. Furthermore, this allows to track known security vulnerabilities introduced by such dependencies, as well as export restrictions and other important metadata.

The management of FLOSS dependencies has grown in an ad-hoc fashion in most companies. The architectural models used for this are often designed to satisfy a managerial perspective and neglect more fine-grained issues such as compliance information of bill of materials (BOM) details. As a consequence, there is a mismatch between an architectural model that supports the management of FLOSS dependencies and the models we found in industry. This leads to costly integration operations to ensure license compliance. Many companies develop own tooling to merge documented license information and generate more comprehensive reports about the use of FLOSS in their products. The lack of consistent concepts in the underlining models also causes manual intervention in the dependency management and license compliance processes, which slows down production. The vast, and still increasing, amount of the commercially used open source components additionally complicates the process of license compliance.

To address this industry-relevant yet underresearched issue, we performed exploratory single-case study research at one large established software vendor. We wanted to identify the challenges companies face regarding software dependency documentation in the context of FLOSS license compliance and tracking. We also studied the ad-hoc solutions to these challenges in the context of the studied company. The main contribution of our paper is a systematic analysis of the challenges related to the tracking and documentation of open source components and dependencies in companies. As a result of our study, the following categories of challenges coupled with their point solutions emerged:

- Data Gathering for the Compliance Process
- Usage of FLOSS in Products
- Custom Reports
- SPDX Support
- A Central System to Manage Products.

For each category, we report the current issues discovered in the course of the case study, as well as the solutions found at the studied company supported by the analyzed qualitative data through interview quotes and observatory descriptions. We suggest that the identified challenges and solutions can fit together into a conceptual model that we tentatively call the product (architecture) model, which we will investigate and evaluate in further research aiming for broader generalizability beyond this single-case case study.

2 Related Work

In previous work, we reported our findings on industry requirements for FLOSS governance tools [10]. We found a hierarchical list of requirements that indicated the following four key categories: Tracking and reuse of FLOSS components, License compliance of FLOSS components, Search and selection of FLOSS components, and Other requirements (security, education, etc.). We then extended our findings in a subsequent journal article [11] with a new round of qualitative data analysis using five additional interviews, which added the fifth category to the proposed theory focusing on the architecture model for software products.

Among other requirements for tooling, it showed that companies develop their own tools to incorporate FLOSS governance relevant information with architectural information to manage software products. In this paper, we follow up on the topic of software dependency documentation in the context of FLOSS compliance, and focus on the challenges companies face in doing this. We did not identify any literature focusing on this specific topic, but found literature on FLOSS governance and compliance more broadly.

Hammouda et al. [8] introduced open source legality patterns, which help with architectural design decisions motivated by legal concerns associated with FLOSS licenses. They focus on how the interaction between different components can be managed so that the overall product contains no license violations. The patterns are grouped in the following categories: Interaction legality patterns, Isolation legality patterns, and Licensing legality patterns. An example of an Interaction legality pattern is to switch from static to dynamic linking if a proprietary component depends on a strong copyleft licensed component.

Fendt et al. [3, 2] conducted practitioner-driven research on FLOSS governance, compliance, and policy. In 2016 [3], the authors suggested processes for the successful management of FLOSS use in companies that want to avoid the related legal risks. The proposed processes were designed to serve as best practices and a basis for corporate governance, strategy, policy, and process implementation. In 2019 [2], Fendt et al. followed up with an experience report on their use of open source tools and services for open source license compliance. They used several well-known tools such as SW360[1] and Fossology[2], integrated in a complete compliance toolchain.

From a model perspective, the Software Package Data Exchange (SPDX$^{\mathrm{TM}}$) specification is the de-facto standard to exchange the bill-of-materials of a product. The goal of the SPDX specification is to enable companies and other organizations to share license and component metadata for a software package. This specification was the result of a shared effort from different organizations to address the needs of various participants in software supply chains [17]. The SPDX License List is a key factor for the adoption of the SPDX standard in companies [5]. This curated list contains the commonly used open source licenses and enables open source license identification through a unique identifier.

German et al. [6] described a method to model dependencies that are required to build and execute a program. The classification criteria for dependencies in this model contain information about a dependency's type (explicit or abstract), if it is optional, in which stage it is required (build-time, run-time, etc.), and the usage method (stand alone, library, and more). The packages in this model are also aware of licenses. Additionally, they suggested a method to visualize the dependency graph of a software package based on their model. It also allowed showing inconsistencies, such as license conflicts between packages. For their analysis, they used packages of a FLOSS distribution (Debian 4.0) and demonstrated how the model could help provide insights into the FLOSS ecosystem.

[1] https://projects.eclipse.org/proposals/sw360
[2] https://www.fossology.org

3 Research Methods

Given the unexplored nature and the practical relevance of the research topic, we conducted a single-case case study informed by Yin [18] to study the software dependency documentation in terms of FLOSS license compliance and tracking. We chose the case study methodology over alternatives (e.g. grounded theory, experiments) because of its suitability for the emerging and complex phenomena that can be best studied in their practical contexts [18]. Given the complexities of the corporate FLOSS use, governance, and compliance in general [9] and of the software dependency documentation issues in particular [11], we chose a case study company that actively uses open source components in products, while encountering issues in tracking and documenting this use as part of the complete product architecture.

We chose a large multinational enterprise software company with mature FLOSS awareness and use. The company provides services to retail and enterprise customers, and distribute open source and closed source software to them as well. In our search for the appropriate case study company, we leveraged our professional network and a German industry-academia collaboration project that two of the co-authors were part of Software Campus[3]. This enabled us to find a company that is a heavy open source software user at a certain FLOSS governance maturity level where the basics are covered but more advanced aspects of such governance are in flux. The latter included our focal research topic – software dependency documentation in terms of FLOSS license compliance and tracking. The company and the interview data are anonymized as per the company's request.

We outlined in the case study protocol [18] that our case study was both descriptive and explanatory. It was descriptive in detailing reports of what the current state of open source governance at the studied company was when it came to the software dependency documentation focused on FLOSS license compliance and tracking. It was explanatory in presenting some reasons why certain issues in product architecture and open source software dependency documentation arise. When addressing the descriptive side of our study the main co-author visited the case study company for direct observation of the current practices in their real-life context. As for the explanatory aspect, we conducted interviews with five employees at the case study company to cover the breadth of issues faced by the people in different roles responsible for product architecture and FLOSS tracking and documentation. We interviewed employees with different views on the usage of FLOSS within the company. The first two were FLOSS compliance managers. The third person was a product owner and covered the role of a user of open source. The next person was a product manager. The last person was a developer and in his role responsible for the company's architecture model which was used for FLOSS compliance.

To analyze the conducted interviews, we employed computer-assisted qualitative data analysis (QDA) software (CAQDAS) to ensure the systematic analysis

[3] https://softwarecampus.de/

of the data and the traceability of our theory to the data. Informed by Jansen's logic of qualitative survey research [12], we conducted an open (inductive) survey, in which the relevant topics of product architecture and FLOSS tracking were identified through the interpretation of raw data – employee interview transcripts. This approach was in contrast to the other type of qualitative surveys – the pre-structured survey, in which some main topics, dimensions, and categories would be defined before the study, which was not suitable for our exploratory study on this emerging domain.

4 Results

In this section, we present our results on the challenges of tracking and documenting open source dependencies in products.

As Fendt and Jäger [2] show, the required process to be compliant is more complex than just running a license scanner tool on the codebase. Instead, it should be embedded into a bigger process of FLOSS management. Even a company with mature FLOSS awareness and use does not always have a good solution in place to overcome all the challenges of FLOSS dependency management. This leads to workarounds which often force a manual treatment.

In the following, we describe our findings and discuss their implications on the whole FLOSS license compliance process. The main challenges we found are:

– Data Gathering for the Compliance Process
– Usage of FLOSS in Products
– Custom Reports
– SPDX Support
– A Central System to Manage Products.

Data Gathering for the Compliance Process If a developer wants to use a FLOSS component it needs to be approved by the company's open source software office. FLOSS compliance processes are in place to manage the approval of a FLOSS component use. Fendt et al. [3] describe that a FLOSS management process inside their company consists of several phases. The first phase is the Request Phase, in which development teams initiate an approval request for a new FLOSS component they would like to use. The FLOSS compliance process of our case study company matches this first phase by requesting the use of a FLOSS component. The usage of only approved FLOSS components helps avoid legal compliance risks introduced by unknown components. In our previous studies about industry requirements on FLOSS governance and compliance tools, we found that companies want to have an automated process of adding new FLOSS components and their metadata into a common architectural model [11], captured in the following requirement:

"The tool should allow automated adding of FLOSS components and their metadata into the repository using the product architecture model" – Requirement 5.b. [11]

But in reality, users of FLOSS components often have to provide information about a component by hand. This manual step of collecting information is time-consuming and can slow down the whole approval process. The typical required information for an approval decision includes:

- Licenses and copyright information
- Project references, like the homepage URL
- Uploaded source code and binaries, or provide location to download them
- Clarification if component include cryptographic functionalities
- Export Control Classification Number (ECCN)[4]
- Does the component require a specific runtime, like Java Runtime Environment (JRE).

As a key challenge at the studied company, we find that if a developer is uncertain whether the component in a specific version is already approved, he has to use a similar process by entering all the information about the component into the system used for the approval process. This is time-consuming, according to an interviewee from the case study company:

“It’s very time-consuming to collect all the necessary information to provide the [FLOSS approval] request” – Product owner (user of open source)

Usage of FLOSS in Products Whether a FLOSS component and its license are suitable for a company’s product also depends on how the product is to be used. At our studied company they distinguish between the following four usage types: 1) on-premise installation on a client’s machine; 2) providing services as a cloud solution, aka. software-as-a-service (SaaS); 3) used as a library to be integrated into other products; 4) internal use only. These usage types are not exclusive and can be combined, e.g. the product is provided as a cloud service and is only available for internal use, which increases the complexity of the underlining architectural model.

One interviewee explained that, once a FLOSS component is listed in the catalog of approved components, every developer can use these components in the limits of the usage type. For example, the Microsoft SQL Server (with an on-premises license purchased without software assurance and mobility right)[5] can be bundled and shipped with a product, but you’re not allowed to use it in a product that is a cloud service (SaaS).

For a different usage type, FLOSS components used in a product that is only for internal use do not have to go through a costly clarification process. This comes with the fact that internally used software is not distributed and as such, there is no need to ensure license compliance.

”You can download everything, but once it goes into the product that’s delivered to customer, you have to ask: Can you use it?” – FLOSS compliance manager

[4] https://www.bis.doc.gov/index.php/licensing/commerce-control-list-classification/export-control-classification-number-eccn

[5] https://www.microsoft.com/en-us/licensing/news/updated-licensing-rights-for-dedicated-cloud

Custom Reports We found that at our case study company the management demanded reports for their products which included information about the incorporated FLOSS components. This helped track FLOSS assets and assisted in FLOSS risk management.

To provide reports of FLOSS usage in products, license information has to be combined with the architectural documentation of a product. For that, companies develop their own tools which combine all the necessary information from different sources.

One particular challenge here is to keep reliable references between a FLOSS component, their artifacts, and related data. Inconsistencies in these references lead to manual data clean up work. For example, a scan for FLOSS components would identify multiple already known components but couldn't match them with data from a central FLOSS management system.

"This [specific] report tries to figure out where in our vault is the third part component and it matches with my product. Quite often it happens that it doesn't match even if I have provided the very similar and identical source and binaries." – Product owner (user of open source)

A combined model could help simplify the creation of reports and avoid mismatches of components by inconsistent references.

SPDX Support SPDX is the defacto standard to exchange license compliance information. This is especially useful within software supply chains, where suppliers are often expected to provide SPDX documents [9] alongside the delivered software. Many license compliance tools support SPDX as an exchange format for license compliance information and other metadata. Therefore, companies develop custom tools to be able to consume and produce SPDX documents.

At the case study company, we see the benefits of SPDX and integrated support in their toolchain for it. While SPDX standardizes the exchange of license information, a valid SPDX document can lack information and thus not ensure a complete BOM representation. When the studied company required the BOM from a supplier both parties had to agree on how certain information would be stored in the SPDX document.

"But we learnt that SPDX has no clear prescription, what is required, and what's not, to give you, you can do this and that, and we ended up adding a few own fields for those information we think we need." – FLOSS compliance manager

A Central System to Manage Products One interview partner reported that they used a central system to manage their products in which they could incorporate all the information needed for reports and license compliance artifacts. While our interview partner saw high value in having a central system, it did not circumvent all the challenges of license compliance. For example, they reported that relationships between components of a product were represented only as simple dependencies. This simplification produced a high-level view on the composition of a product, which was not enough to generate precise reports on the full product architecture.

"[A central system] is for me not enough [...] we have all the data but the relations between things [are insufficient]" – Product manager

As described before, data gathering from different tools and systems to feed the central system tends to be error-prone. A central system to manage products often grew over time from a system that was not designed to ensure license compliance in the first place. Therefore, some aspects of license compliance result in costly operations, which constitutes another major challenge we discovered during our case study.

5 Discussion

The previous section presents five major challenges the analyzed company has regarding their FLOSS component management processes. In the following paragraphs we present the implications of our findings.

Coping with Amount of FLOSS Dependencies One factor that makes the described challenges even more difficult is the sheer quantity of FLOSS dependencies. FLOSS itself is often built on top of other FLOSS, which results in a snowball effect of dependencies. According to a report [14] from 2016, the average commercial product 80% of the code comes from FLOSS. Our interview partners also described that the sheer amount of FLOSS in their products leads to a time-consuming FLOSS license compliance process. To cope with the increasing amount of FLOSS dependencies, the aim is to automate the license compliance process as much as possible.

Considering Product Shipping Information Because the architecture documentation was initially not designed with compliance information in mind, it is difficult to deal with FLOSS in products. As described, costly operations are required to collect and maintain license compliance information. Another effect is that **the documentation of a product does not represent the actual shipped product**. An example of such a divergence is when a product's feature depends on the purchased product key, like a home vs a professional version of a product. While the product is still the same, the shipped version of the product differs in accessible components. More detailed documentation on how a product is used could allow deriving a BOM which only covers all necessary components of the actual shipped product.

Describing Inter-Dependency Relationships We found that products and their dependencies on FLOSS components and infrastructure are represented in a simplified manner. Besides issues on representing a product in its shipped version, FLOSS licenses may have different obligations dependent on how the component is incorporated in a product. For example, the LGPL license family distinguishes between static and dynamic linking for applying the viral copyleft effect. Information about required runtime environments, which also may introduce additional dependencies, are hard to represent next to architectural and license relevant information. Nevertheless, the studied company relies on this information and has to put additional effort into developing and maintaining custom tools to generate accurate reports and artifacts like the BOM.

Existence of an Underlying Conceptual Model Regarding an architectural model, FLOSS license information must be incorporated in an unambiguous fashion. For example, using unique SPDX license identifier instead of unstructured text to describe the declared license. Our previous studies cover the requirements for automation of the FLOSS compliance process, but those requirements are not fulfilled by tools to the extent which is required [10, 11]. An elaborate representation of inter-dependency relationships should be an inherent part of the architectural model. Also, information for shipping the product as mentioned above should be included in this model. The need for automation and the resulting interoperability of tools suggests there is a need for a general underlying conceptual model representing the product. The consensus on such a conceptual model may also help to avoid the development of company-specific tooling solutions.

6 Limitations

Since we investigated the challenges of FLOSS dependency documentation and tracking in the context of only one company, we do not claim generalizability for the findings. Acknowledging this common limitation of single-case case studies, we were nonetheless able to deeply investigate this emerging research topic identifying five advanced subtopics that went beyond the previous research on FLOSS license compliance and dependencies. Moreover, to mitigate the anticipated limitation, we were careful to choose a representative company with some familiarity of using open source components in products, but limited awareness when it came specifically to FLOSS component tracking and documentation.

7 Conclusion

To study the emerging issue of tracking FLOSS components and dependencies in software product architectures, we performed a case study in a multinational enterprise software company as part of an industry-academia collaboration project. In this exploratory study, we conducted and analyzed five interviews with key stakeholders of FLOSS component tracking and documentation in the studied company.

We identified the core challenges of managing FLOSS dependencies and their integration with existing product architecture infrastructures within a company context. We discussed in detail the challenges related to the data gathering for the compliance process, FLOSS usage in products, custom reports, SPDX support, and to the centralized system for managing products.

Our study suggests that underlying the various point solutions that we found at this vendor lies a conceptual model that we tentatively call the product (architecture) model. In future cross-vendor work, we will investigate whether this conceptual model can be expanded to become a unifying model for all open source dependency management.

Acknowledgements We thank Sebastian Schmid for his generous feedback that helped us improve the paper significantly. We also thank our industry partners that provided their valuable time and expertise for this research project. This research was funded by BMBFs (Federal Ministry of Education and Research) Software Campus 2.0 project (OSGOV, 01IS17045).

References

1. Deshpande, A., Riehle, D.: The total growth of open source. In: IFIP International Federation for Information Processing, vol. 275, pp. 197–209. Springer US, Boston, MA (2008). https://doi.org/10.1007/978-0-387-09684-1_16
2. Fendt, O., Jaeger, M.: Open source for open source license compliance. In: Bordeleau, F., Sillitti, A., Meirelles, P., Lenarduzzi, V. (eds.) Open Source Systems. pp. 133–138. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-20883-7_12
3. Fendt, O., Jaeger, M., Serrano, R.J.: Industrial Experience with Open Source Software Process Management. Proceedings - International Computer Software and Applications Conference **2**, 180–185 (2016). https://doi.org/10.1109/COMPSAC.2016.138
4. Fitzgerald: The Transformation of Open Source Software. MIS Quarterly **30**(3), 587 (2006). https://doi.org/10.2307/25148740, http://www.jstor.org/stable/10.2307/25148740
5. Gandhi, R., Germonprez, M., Link, G.J.: Open data standards for open source software risk management routines: An examination of SPDX. Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work pp. 219–229 (2018). https://doi.org/10.1145/3148330.3148333
6. German, D.M., González-Barahona, J.M., Robles, G.: A model to understand the building and running inter-dependencies of software. Proceedings - Working Conference on Reverse Engineering, WCRE pp. 140–149 (2007). https://doi.org/10.1109/WCRE.2007.5
7. Hammond, J., Santinelli, P., Billings, J.J., Ledingham, B.: The tenth annual future of open source survey. Black Duck Software (2016)
8. Hammouda, I., Mikkonen, T., Oksanen, V., Jaaksi, A.: Open source legality patterns: Architectural design decisions motivated by legal concerns. Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments, MindTrek 2010 pp. 207–214 (2010). https://doi.org/10.1145/1930488.1930533
9. Harutyunyan, N.: Corporate Open Source Governance of Software Supply Chains. doctoralthesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) (2019)

10. Harutyunyan, N., Bauer, A., Riehle, D.: Understanding industry requirements for FLOSS governance tools. In: IFIP Advances in Information and Communication Technology. vol. 525, pp. 151–167. Springer, Cham (jun 2018). https://doi.org/10.1007/978-3-319-92375-8_13, http://link.springer.com/10.1007/978-3-319-92375-8_13
11. Harutyunyan, N., Bauer, A., Riehle, D.: Industry requirements for FLOSS governance tools to facilitate the use of open source software in commercial products. Journal of Systems and Software **158**, 110390 (2019). https://doi.org/10.1016/j.jss.2019.08.001
12. Jansen, H.: The logic of qualitative survey research and its position in the field of social research methods. In: Forum Qualitative Sozialforschung/Forum: Qualitative Social Research. vol. 11 (2010)
13. von Krogh, G., von Hippel, E.: The Promise of Research on Open Source Software. Management Science **52**(7), 975–983 (jul 2006). https://doi.org/10.1287/mnsc.1060.0560, http://pubsonline.informs.org/doi/abs/10.1287/mnsc.1060.0560
14. Pittenger, M.: Open source security analysis: The state of open source security in commercial applications. Black Duck Software, Tech. Rep (2016)
15. Riehle, D.: The economic motivation of open source software: Stakeholder perspectives. Computer **40**(4), 25–32 (apr 2007). https://doi.org/10.1109/MC.2007.147, http://ieeexplore.ieee.org/document/4160218/
16. Riehle, D.: The commercial open source business model. In: Lecture Notes in Business Information Processing. vol. 36 LNBIP, pp. 18–30 (2009). https://doi.org/10.1007/978-3-642-03132-8_2, http://link.springer.com/10.1007/978-3-642-03132-8_2
17. Stewart, K., Odence, P., Rockett, E.: Software Package Data Exchange (SPDX^{TM}) Specification. International Free and Open Source Software Law Review **2**(2), 191–196 (2012). https://doi.org/10.5033/ifosslr.v4i1.45, https://spdx.org/
18. Yin, R.K.: Case study research and applications: Design and methods. Sage publications (2017)