The Journal of Systems and Software 158 (2019) 110390

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Industry requirements for FLOSS governance tools to facilitate the use of open source software in commercial products

Nikolay Harutyunyan*, Andreas Bauer, Dirk Riehle

Friedrich-Alexander University Erlangen-Nuernberg, Erlangen 91058, Germany

ARTICLE INFO

Article history:
Received 15 November 2018
Revised 2 July 2019
Accepted 2 August 2019
Available online 2 August 2019

2020 MSC:
00-01
99-00

Keywords:
Open source software
FLOSS
FOSS
Open source governance
FLOSS governance tools
Company requirements for FLOSS tools

ABSTRACT

Virtually all software products incorporate free/libre and open source software (FLOSS) components. However, ungoverned use of FLOSS components can result in legal and financial risks, and risks to a firm's intellectual property. To avoid these risks, companies must govern their FLOSS use through open source governance processes and by following industry best practices. A particular challenge is license compliance. To manage the complexity of governance and compliance, companies should use tools and well-defined processes. This paper investigates and presents industry requirements for FLOSS governance tools, followed by an evaluation of the suggested requirements.

We chose eleven companies with an advanced understanding of open source governance and interviewed their FLOSS governance experts to derive a theory of industry requirements for tooling. We extended our previous work adding the requirement category on the architecture model for software products.

We then analyzed the features of leading governance tools and used this analysis to evaluate two categories of our theory: FLOSS license scanning and FLOSS components in product bills of materials. The result is a list of FLOSS governance requirements. For practical relevance, we cast our theory as a requirements specification for FLOSS governance tools.

1. Introduction

In recent years more and more companies have been using open source components into their products with an estimate of 95% of all commercial software including open source software (Franch Gutiérrez et al., 2013). Companies increasingly realize the benefits of using FLOSS components in their products, going beyond the commonplace use of FLOSS development tools (Deshpande and Riehle, 2008; Fitzgerald, 2006; von Krogh and von Hippel, 2006; Riehle, 2007; 2009). However, they need to govern and regulate their use of FLOSS components to avoid common threats, such as FLOSS license non-compliance leading to copyright and patent infringement, which can result in litigation, cease and desist claims or product recalls (Software, 2017; Radcliffe and Odence, 2017; Riehle and Lempetzeder, 2014; Ruffin and Ebert, 2004). In the context of this paper, we define FLOSS governance as the set of processes, best practices, and tools employed by companies to use FLOSS components as part of their commercial products while minimizing their risks and maximizing their benefit from such use.

FLOSS governance processes and tools can apply to the commercial use, contribution or leadership of FLOSS projects. We limited the scope of this paper to the commercial use of FLOSS components, intentionally excluding governance considerations of FLOSS contribution or leadership by companies. This is in line with our earlier definition of FLOSS governance. Such focus allowed us to generate an in-depth theory covering the industry involvement with open source that is of highest practical relevance to most companies today and novel to FLOSS research (Hauge et al., 2010).

Despite the practical relevance of the issue, research has been slow to address the use of FLOSS in products. The existing literature is limited to general FLOSS governance research (Aksulu and Wade, 2010; Bonaccorsi and Rossi, 2003; Capra et al., 2008), to the research of the governance of FLOSS communities and their development practices (De Laat, 2007; Lattemann and Stieglitz, 2005; Riehle, 2011; Sadowski et al., 2008), and to FLOSS license compliance related governance (Jaeger, 2017; Gangadharan et al., 2008; 2012; German et al., 2010b; 2010a; German and Hassan, 2009; Di Penta et al., 2010; Stewart et al., 2012; Wang and Wang, 2001). However, past research has not comprehensively addressed FLOSS governance requirements and best practices in the industry.

* Corresponding author.
E-mail addresses: nikolay.harutyunyan@fau.de (N. Harutyunyan), andi.bauer@fau.de (A. Bauer), dirk@riehle.org (D. Riehle).

https://doi.org/10.1016/j.jss.2019.08.001
0164-1212/

A particularly practical aspect of FLOSS governance is its automation through tooling, which ensures increased efficiency and better integration into the development process. Companies need tools to scan all the used open source files, because manual checks are time consuming and virtually impossible for large systems. When talking about tools, we consider requirements in the context of the both the explicit open source use (e.g. with SPDX license declarations), and implicit one (e.g. unstructured license statements). In our study we asked the following research question:

RQ: What are the core industry requirements for FLOSS governance tools needed to facilitate the use of FLOSS components in commercial products?

The research method employed is an adaptation of the grounded theory method (Charmaz, 2014; Corbin and Strauss, 2014) called the QDAcityRE method for structural domain modeling using qualitative data analysis (Kaufmann and Riehle, 2019). We chose this novel, yet promising research method because it enables using qualitative data analysis (QDA) to develop a theory that can be specifically cast as a requirements specification. Answering our research question, we aimed to cast our theory as a list of common industry requirements for FLOSS governance tools. This format is well-understood in the industry and can, therefore, ensure a high practical value of our research results. Data gathering and analysis were performed using formal semi-structured interviews, researcher notes, and materials review. We interviewed 20 FLOSS governance and compliance experts from 11 diverse companies chosen through a theoretical sampling of more than 140 companies.

There are few reports on the commercial adoption of FLOSS that are cast as lists of requirements focusing on technical and managerial aspects of using FLOSS in proprietary products (Wang and Wang, 2001). However, neither academic nor practitioner literature offers a detailed list of industry requirements for FLOSS governance or its tooling that goes beyond a high-level of abstraction (Ope, 2019). In this paper, we extended our previous research on the topic (Harutyunyan et al., 2018), addressing this research gap with our main contribution the theory of industry requirements for FLOSS governance tools. Our theory indicated five key categories of FLOSS governance tool requirements in no particular order:

- Tracking and reuse of FLOSS components
- License compliance of FLOSS components
- Search and selection of FLOSS components
- Architecture model for software products
- Other requirements (security, export restrictions etc.).

We then broke down each of these categories into detailed requirements and sub-requirements. These requirements are presented in the Tables 3–6.

Extending our previous research (Harutyunyan et al., 2018), we added one company (Company 11) to our sample and interviewed 5 open source governance experts from Company 11. Adding the data into our qualitative data analysis, we derived an additional category of industry requirements for open source tooling, focused on the architecture model for software products. The latter is a novel contribution of this paper.

To evaluate our theory, we analyzed marketing materials and demos of 6 widely used and representative FLOSS governance tools. We compared the key tool features with our suggested theory and evaluated our proposed requirements confirming many of them. In future publications, we also plan to address other aspects of FLOSS governance in detail, including industry best practices for FLOSS supply chain management and license compliance.

This paper is structured as follows. Section 2 gives an overview of related work detailing prior open source governance research, FLOSS governance tooling, and industry requirements for governance tools. Section 3 outlines our research method for conducting and analyzing expert interview in ten leading companies in terms of open source governance. Section 4 presents our research results including a theory of industry requirements for open source governance tools cast as a detailed list of requirements, specific requirement descriptions, and corresponding traces in our data. Section 5 describes the evaluation of our theory. Section 6 discusses the implications of our findings and presents questions for further research. Section 7 goes on to present the limitations of our study. Section 8 concludes the paper.

2. Related work

Early research on FLOSS governance in companies was part of the broader research on the commercial use of FLOSS development tools and components (Aksulu and Wade, 2010; Hauge et al., 2010; Höst and Oručević-Alagić, 2011). In a systematic literature review on FLOSS adoption in industry, Hauge et al. (Hauge et al., 2010) identified a limited amount of research focusing on FLOSS component selection by companies (Cruz et al., 2006; Deprez and Alexandre, 2008; Hummel et al., 2008; Umarji and Sim, 2014) and knowledge sharing within FLOSS communities (von Krogh et al., 2005; Lakhani and Von Hippel, 2003; Sowe et al., 2008). They did not identify any academic studies focused on the actual industry practice of using FLOSS components in products, thus suggesting that further research is needed on this topic. Our literature review confirmed this research gap prompting us to conduct this study of 11 industry-representative companies.

We set our research scope and that of the related work review to the commercial use of FLOSS components in products and industry requirements for FLOSS governance tooling. We employed snowballing as a search approach for literature research. We explicitly excluded FLOSS governance related to industry contribution to or leadership of FLOSS projects. We did not identify literature explicitly focused on FLOSS governance tool requirements. However, we found indirect references to the topic that we used as a starting point for our research. We derived three key categories of FLOSS governance requirements that can be addressed through tooling:

- Tracking and Reuse of FLOSS components (Helmreich and Riehle, 2011; Popp, 2015; LF2, 2019)
- License Compliance of FLOSS components (Jaeger, 2017; Gangadharan et al., 2008; 2012; German and Hassan, 2009; German et al., 2010b; 2010a; Di Penta et al., 2010; Stewart et al., 2012; Wang and Wang, 2001; Kapitsaki et al., 2015)
- Search and Selection of FLOSS components (Cruz et al., 2006; Deprez and Alexandre, 2008; Hummel et al., 2008; Semeteys, 2008; Umarji and Sim, 2014)

Tracking and Reuse With the growing availability of high-quality FLOSS components, software developers increasingly use FLOSS components in commercial products. FLOSS governance policies in many companies require developers to track and document such FLOSS use (Helmreich and Riehle, 2011; Popp, 2015). This enables the well-structured management and reuse of FLOSS components that have been added into product software. Umarji and Sim (2014) suggest using FLOSS governance tools to create and maintain libraries of reusable FLOSS components. Our findings confirm this as one of the industry requirements for FLOSS governance tools.

Other requirements focus on supply chain management (Ope, 2019), automated management of bill of materials (Stewart et al., 2012), maintenance of FLOSS component metadata in product architecture models (Riehle and Harutyunyan, 2017), etc. Our theory confirms and captures these requirements.

License Compliance Wang and Wang present a number of requirements for industry adoption of FLOSS. Some of these

requirements can be translated into industry requirements for FLOSS governance tools. The authors suggest a managerial requirement for license compliance that includes understanding different FLOSS licenses and documenting their terms (Wang and Wang, 2001). Our theory suggests that industry requires the use of FLOSS governance tools for documenting company interpretation of most common and used FLOSS licenses and their implications. This requirement is also confirmed by industry associations, such as The Open Source Automation Development Lab eG, which in 2017 attempted to standardize FLOSS license obligations through checklists and own license describing language that can eventually be used in a FLOSS governance tool (Jaeger, 2017).

Other industry requirements for compliance tools include automated FLOSS license scanning (Gangadharan et al., 2012; German and Hassan, 2009; Kapitsaki et al., 2015), automated FLOSS code detection in companys codebase and in its supply chain using source code and binary scans (German et al., 2010a; Di Penta et al., 2010; Stewart et al., 2012), checking FLOSS license compatibility when mixing licenses (German et al., 2010b) etc. We confirm all these requirements through expert interviews and formalize them in our theory, while recognizing the technological complexity of fulfilling these requirements by the currently existing tooling.

Search and selection: Umarji and Sim (2014) surveyed a sample of 69 programmers. Their research suggested that software developers require and use tools for the search and selection of FLOSS components. The majority of the survey respondents said they used general-purpose search engines with some also using project hosting sites and code-specific search engines. Our expert interviews confirmed the requirement for search and selection of FLOSS components. A requirement in our proposed theory formalizes this industry need.

Other industry requirements for search and selection of FLOSS components focus on the automated identification of software families and types of FLOSS communities (Sadowski et al., 2008). Our theory did not confirm the industry requirement for the tool-assisted software family identification, but did confirm the need for the tool-assisted identification and evaluation of FLOSS communities. Many other requirements are suggested in both academic literature and practitioner white papers. However, in this section, we combined and presented the literature related to only several key requirements due to our narrow scope.

Gonzalez-Barahona et al. (2013) studied how companies interact with FLOSS communities by applying data analytic techniques on the software repositories. In our theory, we did not identify industry requirements for FLOSS tools focused on community management or engagement, which can be explained by our focus on the use of open source components in companies, but not on the contribution to FLOSS communities.

Stol and Ali Babar (2010) did a systematically literature review on the challenges in using FLOSS in product development. They identified several studies that reported that organizations have an issue with the complex FLOSS licensing situation and have concerns about intellectual properties and rights. Our requirements on identification and interpretation of open source licenses address this issue.

3. Research method

We conducted a two-step study that consists of:

1. Deriving a theory based on our understanding of key industry requirements for FLOSS governance tools through expert interviews.
2. Evaluating our understanding of industry requirements through marketing materials and demos of existing FLOSS governance tools.

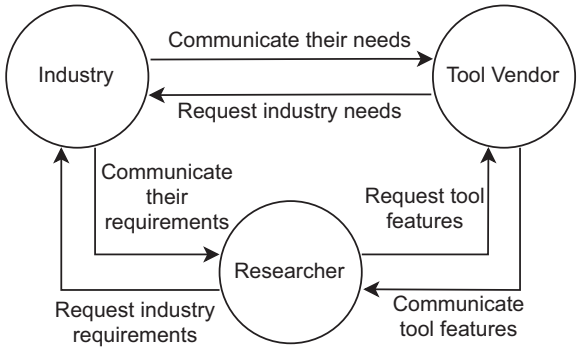

Fig. 1. Theory building using industry requirements and theory evaluation using tool features.

Our research approach is represented in Fig. 1 and explained below.

For theory building, we conducted 20 interviews with eleven leading companies to understand their requirements for FLOSS governance tools. We employed a method for structural domain modeling using qualitative data analysis called QDAcityRE (Kaufmann and Riehle, 2019), a method that builds on GT-based analysis techniques (Charmaz, 2014; Corbin and Strauss, 2014). Corbin and Strauss (2014) define grounded theory as a method that consists of systematic, yet flexible guidelines for collecting and analyzing qualitative data to construct a theory from that data. Kaufmann and Riehle (2019) accept this definition, but extend the method to a more structured, traceable and iterative one providing guidelines for data collection, creation and application of a code system. This enabled us to use the QDAcity-RE method for requirements engineering based on our industry expert interviews. The result is a theory of industry requirements for FLOSS governance tools cast as a requirements specification.

For theory evaluation, we reviewed marketing materials and demos of 6 widely used FLOSS governance tools. We used the QDAcityRE method and qualitative data analysis to derive the common features they offer to meet industry needs for automating FLOSS governance.

Assuming that the tool vendors as a whole understand industry needs and offer tools that address these needs, we compared the common tool features to our theory of industry requirements. We evaluated which tool features match the industry requirements in our proposed theory and which ones do not. We used this evaluation to demonstrate that our theory represents the current state of industry requirements for FLOSS governance tools. To the extent that our theory agrees with tool features, we put the work of industry product managers onto a sound scientific base of theory development based on the users perspective.

3.1. Theoretical sampling

For theory building, we chose eleven companies sampled from our industry network of about 140 companies with advanced FLOSS governance practices. The companies in our sample have an advanced understanding of FLOSS governance and use internal and/or external governance tools. We conducted polar theoretical sampling to cover a diverse and representative set of companies. Polar sampling aims to choose companies with highly varying characteristics. We considered diverse dimensions including types of business models, customer types, company size, market position, and company maturity. The resulting sample of companies includes small, medium and large companies with both enterprise and retail customers and varying business models. The list of companies

Table 1
Theoretical sample of companies.

Company	Company domain	By business model	By type of customer	By size (employees)
Company 1	Consulting	SP-OS, SDS	Enterprise	Medium
Company 2	Automotive	SDS	Enterprise	Small
Company 3	Automotive	SDS	Enterprise	Large
Company 4	Enterprise Software	SP-OS	Enterprise, retail	Medium
Company 5	Enterprise Software	SP-CS	Enterprise, retail	Medium
Company 6	Enterprise Software	SP-OS, SP-CS, OP, GT	Enterprise, retail	Large
Company 7	Enterprise Software	SP-OS, MC, GT	Enterprise, retail	Medium
Company 8	FLOSS Foundation	OSF	Enterprise, retail	Small
Company 9	Hardware and Software	OP	Enterprise	Large
Company 10	Legal	MC	Enterprise, government	Large
Company 11	Enterprise Software	SP-OS, SP-CS, MC, GT	Enterprise, retail	Large

Table Legend: SDS = Software development service, SP-OS = Software product vendor for open source software, SP-CS = Software product vendor for closed source software, GT = Governance tool providers, MC = Management consulting, OSF = Open source foundation, OP = Other products incorporating software.

Table 2
Sample of governance tools.

Tool	License	Delivery model	Scannable artifacts	Automation & DevOps integration	Maturity level
Black Duck Hub	Proprietary	CB	Source, binary	DI, AU	Mature
DejaCode	Apache 2.0	CB, OP	Source, binary	ADA	Mature
FOSSology	GPL-2.0	OP	Source, binary	MDA	Mature
FOSSA	Proprietary	CB, OP	Source	DI, AU	Mature
OSS Review Toolkit	Apache 2.0	OP	Source	MDA	Newer project
WhiteSource	Proprietary	CB, OP	Source, binary	DI, AU	Mature

Delivery model: CB=Cloud-based, OP=On premise.
Automation & DevOps integration: DI=Dev. integration provided, AU=automation provided, ADA=Dev. integration and automation as an additional service, MDA=Manual Dev. integration and automation required.

and their essential characteristics are presented in Table 1. Company names are anonymized per their request.

For theory evaluation, we chose 6 widely used and prominent FLOSS governance tools that represent the broader spectrum of FLOSS governance tools (LF2, 2019). Not all tools compete but have some overlap in their functionalities, like support for license scanning or component repository management. To reduce bias, we made sure that our selection differs in these dimensions:

- By the **license** under which a vendor makes its tool available. The sampling contains tools that are licensed under permissive and copyleft type open source licenses, and proprietary closed source licenses.
- By the **delivery model** of a tool. A critical factor for companies is the ability to choose whether a software tool is available as a cloud-based service or can be used on-premise, depending on aspects like costs, customization, and security.
- By the **scannable artifacts**. For scanning of license information, tools can analyze source code or binary artifacts. Scanning of binary artifacts is necessary if the source code of dependent components is not available. In contrast scanning of source code artifacts provide better results.
- By **automation and DevOps integration**. Some tools provide plugins to integrate the tools better or easily in a development environment, or working within an automated process, as a continuous integration process. On the other hand, tools which require a manual integration setup are often more suitable for a firm's own needs.
- By **maturity level**. In this sampling 5 of 6 tools are established tools in this field and are well known. The OSS-Review-Toolkit is the only exception here. It's a project started in 2017 which has recently gained popularity.

The list of tools and their key characteristics are presented in Table 2. All tools were evaluated on March 2018.

For data gathering, we used semi-structured interviews conducted by one or two researchers with FLOSS governance experts or responsible coworkers from the sampled companies. In seven companies we interviewed one expert, in one company we interviewed two experts, and in two companies we interviewed three experts. In total, we conducted 20 interviews. When possible, we recorded and transcribed the interviews. In three cases we took notes.

We developed key questions and an interview guideline for the semi-structured interviews and kept them stable, except for a few iterative adjustments from company to company, throughout the whole data gathering process. The interviews were exploratory in line with our grounded-theory-based research method.

For data analysis, we followed the QDAcity-RE method, performing iterative and incremental qualitative data analysis (QDA) supported by the MaxQDA software. We developed two separate coding systems for the theory-building using expert interviews and for the theory evaluation using tool marketing materials and demos. During the QDA coding process, we iteratively refined the code system. Upon reaching theoretical saturation (Kaufmann and Riehle, 2019), the code system became the basis for our theory. Individual codes correspond to low-level tool requirements in our requirements specification. Both for theory building and evaluation, our code systems consist of hierarchical codes. We did not apply the top category codes in our QDA. We followed the QDAcity-RE methods QDA process as follows:

- *Open coding*. We created a basic set of codes from which the hierarchy is built. Open codes are direct annotations of primary materials and link to them for data-theory traceability.
- *Axial coding*. We built a code system by deriving more abstract concepts and categories from open codes, thus developing the axes of the code system.
- *Selective coding*. We applied the codes to the gathered data and chose which codes are important and which are not. We adjusted the coding system by removing the irrelevant codes and by adding the ones that emerged when applying the axial codes.

4. Research results

This section presents our theory of industry requirements for FLOSS governance tools, followed by the evaluation of the suggested theory through feature analysis of existing FLOSS governance tools.

We limited our scope to FLOSS governance tools related to the commercial use of FLOSS components, explicitly excluding companies contribution to or leadership of FLOSS projects. We only present the requirements that have been directly derived or inferred from our data, thus excluding the ones that have been presented in the literature, but not confirmed by our industry study. The result is a partial theory that covers the key requirement categories and requirements based on our sample. Analyzing 20 expert interviews, researcher notes and company materials, we derived the following high-level industry requirements for FLOSS governance tools:

- Tracking and reuse of FLOSS components
- License compliance of FLOSS components
- Search and selection of FLOSS components
- Architecture model for software products
- Other requirements (security, export restrictions etc.).

We present each category and detailed requirements below.

4.1. Tracking and reuse of FLOSS components

At its core, FLOSS governance starts from identifying and keeping a record of the open source components used in a companys products. To achieve this, developers need to use various tools to document, track and report their FLOSS use in a systematic and consistent manner. This translates into a number of requirements we identified through our expert interviews. The requirements we grouped under the Tracking and Reuse category cover identifying and reporting the FLOSS use, updating and reusing FLOSS components, as well as maintaining and managing bills-of-materials. We present each of these aspects as follows.

Req 1.1. The tool should help users ***identify the use of FLOSS components in their code base****.*

Companies must identify what open source components end up in their products. Some open source components are added directly by company developers, but others get there through supplied software with or without the knowledge of anyone in the company. In any case, it is the companys responsibility to ensure its products open source license compliance, as well as to check the compatibility with potential export restrictions, security, and quality assurance requirements. Before doing any of these things, companies need to identify exactly what open source components made it into their products, which translates into a requirement to the tools that support this. Talking about their automation efforts in identifying open source components, one expert interviewee from Company 6 says:

"We can automate a lot of things whenever we can track down software, and source code, or binaries, and can identify them, and can say, okay, this is a binary that came from this place under this license. If the information is full, and complete, and correct, that could be automated." — Company 6

Req 1.2. The tool should help users ***report the use of FLOSS components in a product architecture model****.*

Developers use open source components and libraries all the time, but without a defined process or tooling such use is often not reported and not documented. Depending on the open source components license and its use case, this can cause legal and technical issues if discovered. For this reason, companies have processes and tools in place to enable easy reporting of FLOSS components used by developers. Such tools can be integrated into the development toolchain, if a company has advanced open source governance tooling. If not, this can be achieved through basic component repositories or documents for reporting such use. This translates into the requirement to have tools that help report FLOSS components used and their interdependencies in the software architecture. Interviewees from Company 3 share their view on this requirement:

"We have started last year with this repository [tool for reporting open source use]. It's under construction I would say. It's a document folder, like a folder on SharePoint where you can report your open source use - we have an [automated] template for this. It's called a solution blueprint." — Company 3

Req 1.3. The tool should help users ***update FLOSS components and their metadata****.*

Merely identifying and reporting the use of open source components in company products is not enough for the complete open source governance and compliance. Companies must track and regularly update used open source components and their metadata. Failing to do so can cause a number of risks, such as exposure to security vulnerabilities discovered in an old version of an open source component (often newer versions fix the discovered vulnerabilities). To address this issue, companies aim to rely on open source governance tools to efficiently update their open source components and related metadata.

Req 1.4. The tool should help users ***maintain a bill of materials of the FLOSS components used in a product****.*

All companies should track their use of FLOSS components in order to efficiently manage FLOSS integration into their products, as well as to enable the cost-saving use of FLOSS components already used by the companys other developers. Efficient FLOSS component management ensures a companys ability to maintain and produce upon customer request an up-to-date bill of materials.

Req 1.5. The tool should help ***users reuse FLOSS components that have already been used in a product****.*

Open source components and libraries are often reused within the company. Once their licenses and corresponding use cases are checked and approved, these components can be used and reused across the company without further checks saving time for both developers and compliance or open source program office. For efficient reuse, companies establish databases or repositories for the approved open source components.

The detailed subcategories of requirements for Tracking and Reuse of FLOSS components are demonstrated in Table 3.

4.2. License compliance of FLOSS components

FLOSS license compliance is a central aspect and key tool requirement category to the companies we studied. Companies strive to automate license compliance, license scanning, and license management. Some companies employ continuous integration/deployment and thus require appropriate license compliance tools that can be integrated into their development process. Tool requirements for license compliance go on to encompass automated license interpretation, license identification and documentation. We present each of these aspects as follows.

Req 2.1. The tool should help users ***interpret open source licenses****.*

For consistent and scalable FLOSS governance, companies need to interpret the common open source licenses for their use cases. License interpretation includes understanding and documenting companys view on the legal and technical obligations caused by

Table 3
Requirement category 1. Tracking and Reuse of FLOSS components requirements.

1. The tool should help users **identify the use of FLOSS components in their code base**.
   a. The tool should allow reading in an existing code base.
   b. The tool should allow automated finding of open source licenses in an existing code base.
   c. The tool should allow automated finding of open source software checked-in and used by a company developer.
   d. The tool should allow automated finding of open source software not checked-in, but used by a company developer.
   e. The tool should allow automated finding of open source software that is part of the supplied proprietary software using commonly accepted data exchange standards (such as SPDX).
   f. The tool should allow automated finding of open source software that is part of the supplied proprietary software using binary or source code scanning.
2. The tool should help users **report the use of FLOSS components in a product architecture model.**
   a. The tool should allow creating a product architecture model to systematically record use of FLOSS components, their metadata and component dependencies.
   b. The tool should allow manual recording of metadata of the used FLOSS components.
   c. The tool should allow confirming the metadata of FLOSS components identified automatically.
   d. The tool should allow modifying the metadata of FLOSS components identified automatically.
   e. The tool should allow removing the metadata of FLOSS components identified automatically.
   f. The tool should allow automated reporting of a newly used FLOSS component within the build process and/or continuous integration process.
   g. The tool should allow reporting undeclared use of FLOSS components and their metadata.
3. The tool should help users **update FLOSS components and their metadata.**
   a. The tool should allow automated updates of FLOSS components to their newest available versions.
   b. The tool should allow to back up the current versions of FLOSS components before updating them.
   c. The tool should allow automated identification of changed metadata including FLOSS component license and copyright information.
   d. The tool should allow automated history recording of FLOSS components and their metadata.
4. The tool should help users **maintain bill of materials of the FLOSS components used in a product.**
   a. The tool should allow creating a formal bill of material using a commonly accepted data exchange standard (such as SPDX).
   b. The tool should allow automated generation of a formal bill of materials using company's product architecture model.
   c. The tool should allow developers to add identified and reported metadata on used FLOSS components into the formal bill of materials.
   d. The tool should allow developers to update the formal bill of materials.
   e. The tool should allow automated generation of a bill of materials instance in a structured textual format.
   f. The tool should allow automated generation of a bill of materials instance in a commonly accepted data exchange standard (such as SPDX) format.
5. The tool should help users reuse **FLOSS components that have already been used in a product.**
   a. The tool should allow creating a centralized and company-wide accessible FLOSS component repository.
   b. The tool should allow automated adding of FLOSS components and their metadata into the repository using the product architecture model.
   c. The tool should allow automated updating of FLOSS components repository using the product architecture model.
   d. The tool should allow all company developers to access the FLOSS components repository.
   e. The tool should allow searching in the FLOSS component repository.
   f. The tool should allow finding the company developers who used an FLOSS component from the repository.

certain open source licenses in relation to different use cases (e.g. internal use only, use for production only, use in products to be distributed). One industry requirement is to have a tool that helps with license interpretation, though its recognized that full automation here is not possible. An expert from Company 2 talks about the tool requirement for support in license interpretation:

"So, the open source handbook doesn't really present rules in a concrete setup, but what it does is it explains all the interpretations of the licenses that we have. We assess licenses with lawyers, with our internal lawyers, and from these license assessments, we determine certain rules for its usage, modification, and contribution. And these rules for the individual licenses are explained in that document. Well, it's a Word document. It has a common structure, yeah. For every license, we have this kind of setup. Many of those issues are just collected in the Wiki-like system. Its not formally that structured." — Company 2

Req 2.2. The tool should help users ***document the identified licenses of the used FLOSS components in the companys open source license repository or license handbook****.*

Companies need to identify and document the open source licenses of their FLOSS components. This translates into one of the essential tool requirements for license scanning, license identification, and documentation. An expert from Company 7 mentions the tool requirement for automating FLOSS license scanning and identification of other FLOSS component metadata:

"Interviewee: We have a full tool-set that goes through and scans the code, that pulls out all the license information, the authorship [copyright] information, and runs that through our process for verification, for compliance, for compatibility and so forth." — Company 7

Req 2.3. The tool should help users ***find and document the unidentified licenses of the used FLOSS components in the companys open source license repository or license handbook****.*

After a license scan, the identified licenses need to be double checked and reviewed to ensure that the license mixtures are compatible and that all the correct licenses have been identified and documented.

Req 2.4. The tool should help users ***approve the use of a FLOSS component in a product based on FLOSS license compliance guidelines****.*

After interpreting open source licenses and identifying their use of FLOSS components, companies need to check if the component currently in use or the potential components correspond to the companys open source governance use guidelines. If so, their use can be approved, and developers can use these components in their projects. Doing manual component approval might work for small projects, but is inefficient in most cases, especially in larger projects. Thus, companies use tools to automate the component approval process, at least to some extent, which translates into the corresponding requirement.

Req 2.5. The tool should help users ***distribute a product that is compliant with the FLOSS licenses of the FLOSS components used in that product****.*

FLOSS governance is critical in software release management. It is an industry best practice to review software products for FLOSS

Table 4
Requirement category 2. License Compliance of FLOSS components requirements.

1. The tool should help users interpret open source licenses.
   a. The tool should allow user to document open source license interpretations using a formal language or notation supported by the tool.
   b. The tool should provide automated standard interpretation of the most common FLOSS licenses in company's license repository or license handbook.
   c. The tool should allow users to modify license interpretation of the most common FLOSS licenses in company's license repository or license handbook.
   d. The tool should allow users to add license interpretation of the FLOSS licenses of the used FLOSS components to company's license repository or license handbook.
   e. The tool should allow users to change license interpretation in the license repository or license handbook.
   f. The tool should allow developers to request license interpretation of a FLOSS license of an FLOSS component s/he wants to use in a product.
   g. The tool should allow open source program office to discuss license interpretation requests.
   h. The tool should allow open source program office to fulfill license interpretation requests.
2. The tool should help users **document the identified licenses of the used FLOSS components in the company's open source license repository or license handbook.**
   a. The tool should allow creating an open source license repository.
   b. The tool should allow developers, lawyers and managers to read the open source license repository.
   c. The tool should allow automated inventorying of known open source licenses from the product architecture model.
   d. The tool should allow users to add new open source licenses into the open source license repository.
   e. The tool should allow users to remove obsolete open source licenses from the open source license repository.
   f. The tool should support the commonly accepted data exchange standards (such as SPDX).
   g. The tool should allow users to search open source license information in the open source license.
3. The tool should help users **find and document the unidentified licenses of the used FLOSS components in company's open source license repository or license handbook.**
   a. The tool should allow software package scanning to find the open source licenses unidentified previously through product architecture model.
   b. The tool should allow source code scanning for the internally developed code to find the origin of used, but unidentified open source code and its license.
   c. The tool should allow source code scanning for the FLOSS components taken from FLOSS projects to find the origin of used, but unidentified open source code and its license.
   d. The tool should allow binary scanning for the FLOSS components that are part of the supplied proprietary software components to find the origin of used, but unidentified open source code and its license.
   e. The tool should allow automated inventorying of the open source licenses identified because of binary and source code scanning.
   f. The tool should allow manual changing the automatically identified open source licenses.
   g. The tool should allow removing the automatically identified open source licenses.
   h. The tool should support binary and source code scanning integration into the build process and/or continuous integration process.
   i. The tool should allow finding and documenting copyright notices, export restriction information and other compliance-related metadata for FLOSS components used in a product.
4. The tool should help users **approve the use of a FLOSS component in a product based on FLOSS license compliance guidelines.**
   a. The tool should allow creating white lists of company-approved FLOSS licenses according to company policy.
   b. The tool should allow creating black lists of company-blocked FLOSS licenses according to company policy.
   c. The tool should allow updating white and black lists of FLOSS licenses.
   d. The tool should allow creating license interpretation-based rules for automated recommendation on component use approval according to company policy.
   e. The tool should allow developers to request approval of FLOSS components with previously unassessed licenses.
   f. The tool should allow lawyers to approve or block use of FLOSS components due to license incompatibility with company policy.
   g. The tool should allow automated recording of FLOSS license approval decisions in company's open source license repository.
5. The tool should help users **distribute a product that is compliant with the FLOSS licenses of the FLOSS components used in that product.**
   a. The tool should allow automated generating of FLOSS license obligations for each product using product architecture model and open source license repository.
   b. The tool should allow automated assignment of tasks that will ensure compliance with FLOSS license obligations.
   c. The tool should allow automated audit of products bill of materials before distribution.
   d. The tool should allow manual audit of products bill of materials before distribution.
   e. The tool should allow adjusting products bill of materials before distribution.

license compliance before distributing them to customers. In this phase, its essential to use tools that help review the final software products, their bills-of-materials, and their fulfillment of FLOSS license obligations. An example obligation is publishing the source code.

The detailed subcategories of requirements for License Compliance of FLOSS components are demonstrated in Table 4.

4.3. Search and selection of FLOSS components

FLOSS governance tools are also used to help developers search and select appropriate FLOSS components for their projects. This includes requirements for searching the web for open source components and for selecting the ones that fit company guidelines best, as well as for estimating the cost of using a selected FLOSS component. We present each of these requirements as follows.

*Req 3.1. The tool should help users **search for FLOSS components**.*

Some companies we studied encourage developers to first search internally for an open source solution that has been used in the past in the company using knowledge sharing tools or databases. This translates into a requirement for tools to help search for open source components, as seen in the interview in Company 2:

"[Talking about the Wiki-like system for knowledge sharing] the point is that developers themselves when they ask questions can go to that page and then search that. Otherwise, they come to their open source compliance manager and have a discussion with him, and he can point out the relevant pages. And the open source compliance managers, they maintain the web pages and so on." — Company 2

*Req 3.2. The tool should help users **select the best FLOSS components**.*

Companies should use FLOSS governance tools to efficiently search and select the right FLOSS components, which translates

Table 5
Requirement category 3. Search and selection of FLOSS components requirements.

1. The tool should help users **search for FLOSS components.**
   a. The tool should allow an automated search of available FLOSS components using publicly available data.
   b. The tool should allow automated comparison of available FLOSS components using publicly available data.
2. The tool should help users **select the best FLOSS components.**
   a. The tool should allow automated health assessment of open source communities using publicly available data.
   b. The tool should allow automated maturity assessment of open source communities using publicly available data.
   c. The tool should allow automated corporate dependence assessment of open source communities using publicly available data.
   d. The tool should allow automated maturity assessment of open source communities using publicly available data.
   e. The tool should allow automated responsiveness assessment of open source communities using publicly available data.
3. The tool should help users estimate the cost of using a FLOSS component.
   a. The tool should allow automated cost estimation of FLOSS component integration and maintenance in a product.
   b. The tool should allow automated risk assessment of FLOSS community discontinuing its development of the FLOSS component and automated cost estimation of internal maintenance of the FLOSS component.
   c. The tool should allow users semi-automated estimation of the benefit of using a FLOSS component compared to proprietary and in-house development alternatives.

into tool requirements on evaluating different component candidates and selecting one. One interviewee talks about the role of tools in FLOSS component selection process:

"Interviewee: When you move on from a strategic decision to component selection with components of open source projects to be used, then we have a process that we require the projects to name all the open source components to assess that they want to use, that they assess the license, that they check the license, and that they document that and that again this assessment is communicated to upper management and signed off that." — Company 2

*Req 3.3. The tool should help users **estimate the cost of using a FLOSS component**.*

Using open source software does not incur any licensing costs, but there are costs that need to be considered when deciding for using a FLOSS component. A company should check the license compliance and quality assurance of an open source component, update and maintain it, and scan it for potential security vulnerabilities. Estimating these costs can affect the decision of selecting a certain FLOSS component.

The detailed subcategories of requirements for Search and Selection of FLOSS components are demonstrated in Table 5.

4.4. Architecture model for software products

One topic emerged from the interviews that is closely related to requirements on FLOSS compliance tools which is about an architecture model to incorporate all collected compliance-relevant data. While there is SPDX to exchange compliance information, there is no common data structure to incorporate data from different tools. Therefore, the companies we interviewed developed their own version of such an architecture model. To investigate what are common requirements for such models we did additional interviews focused on this topic.

This requirements are shown in Table 6.

*Req 4.1. The model should represent **relationship and dependency information**.*

The interviewees described three types of relationships that are needed for a comprehensive dependency mapping. The first type of relationships is between the components of a product to trace back the introduction of dependencies. The second type is about relationships that are arising from infrastructure dependencies, like the Java Runtime Environment (JRE) or a compiler which inject code into the product. The last type of relationships is to external systems. This is useful if a product consumes a service from another system over the network.

"So you have the packaging and the installer also adds some software to it. So the bill of material we added into it is incomplete, because the installer itself is a third party software, and it's taking the package and wrapping it then add scripts." — Company 11

*Req 4.2. The model should represent **license information**.*

The interviewed partners described that having the license information as a simple text is not sufficient enough. A dedicated license model would allow inheriting required information about a license for a component, like the legal duties that come with a license.

Another requirement in this category is that the model should represent license policies which can be applied to a product's components. This way, third party components with not approved licenses can be rejected automatically.

"We create a data model of the license, the necessary metadata. And then this is added to the license, and whenever we see a component that has the same license, it will inherit automatically the license metadata,usage types approvals, the applications, our legal duties, and so forth." — Company 11

*Req 4.3. The model should allow **optimization**.*

To reduce and avoid redundant FLOSS compliance work for already reviewed components the model must be able to identify these components. Therefore, changes in the corresponding artifacts of a component need to be detected. Also, the model should not prevent the tools from being used in an automated process.

*Req 4.4. The model should represent **a products distribution and release information**.*

This subsection presents the requirements related to product distribution an release. One criterion that a model has to represent is if a product is only for internal use or is a product that will be shipped to customers and needed to be approved by the FLOSS compliance office. Another is the type of how a product is distributed (e.g. on-premise, as cloud service) can lead to different license obligations and so it needs to be considered. Additional, information about the release of a product helps to provide special reports for a release version.

*Req 4.5. The model should represent **information about the quality and reliability of compliance relevant data**.*

The interviewees described that they want to know from which sources the third party components are downloaded and how reliable the related metadata are. Factors that needed to be considered here are if a third party developer published to a source repository or was it done by an intermediary, or if a source repository is well known and has a good reputation.

For compliance-relevant metadata, the reliability of such data is helpful for decision making. Not all collected compliance-relevant data are similarly reliable. Some data collection is done through

Table 6
Requirement category 4. Architecture model for software products.

1. The model should represent **relationship and dependency information.**
   a. The model should represent relationships between components.
   b. The model should represent dependencies that are arising from infrastructure.
   c. The model should represent relationship to other systems.
2. The model should represent **license information.**
   a. The model should have a reusable license model.
   b. The model should represent license policies.
3. The model should allow **optimization.**
   a. The model should allow to identify already detected components.
   b. The model should allow automation of the tool.
4. The model should represent **a products distribution and release information.**
   a. The model should represent different distribution (on-premise, cloud service)
   b. The model should distinguish between private/internal or public/external products.
   c. The model should represent release information.
5. The model should represent **information about the quality and reliability of compliance relevant data.**
   a. The model should represent the origin of compliance relevant data.
   b. The model should represent the reliability of compliance relevant data.
6. The model should allow the **integration of additional data.**
   a. The model should allow the integration of different data.
   b. The model should represent various metadata.
   c. The model should represent export and control restrictions.
   d. The model should be extendable.

automated tools without any review of experts, while other data is the result of a proper review process by experts.

*Req 4.6. The model should allow the **integration of additional data**.*

While compliance tools often focus on specific use cases, like license scanning, an architectural model should incorporate data from different tools with different use cases to create a comprehensive representation of a product. The integration of different collected data for the same use case allows to compare results and conclude correct assumptions. The interviewees reported that an inflexible model for their products causes problems for adapting technological changes, e.g. the introduction of container technologies like Docker for delivering a product.

4.5. Other requirements

Beyond the above mentioned three requirement categories, FLOSS governance and compliance tools are used to fulfill many other requirements companies have. Here we present select ones that are not grouped into any category.

*Req 5.1. The tool should help users **detect and prevent security vulnerabilities in products FLOSS components**.*

Companies need to detect and prevent potential security vulnerabilities in open source components used in their products. This is often done at the same stage as open source license scanning. Therefore the industry requirement is to use tools in checking for security vulnerabilities in open source software used. One expert from Company 1 mentions this need:

"It's that all of our work is on infrastructure, is on running a server and not develop. The only area where we really get in touch with software development is when it comes to security management of open source components maybe even for proprietary software. Even though that is ridiculously important, and [its] even more important to actually know which components you have in your system [to check them for security vulnerabilities]. And frankly, a lot of companies do not know that." — Company 1

*Req 5.2. The tool should help users **document and communicate the companys FLOSS governance strategy, policies and best practices**.*

The companies we interviewed have FLOSS governance strategies, policies and best practices. These are documented and shared in the company so that the developers, managers and legal experts follow the same guidelines around FLOSS governance. Tools can be used to document and communicate these guidelines, as mentioned by an expert from Company 6:

"Interviewee: [We use a] Wiki page and store information about the release, and the sign off of them, so we have also a good amount of tooling, if not to say too many different tools that are in this toolchain to upload it and support open source compliance process, and not forget the last time we've seen our dear artists where we have to model the process, how the process works." — Company 6

*Req 5.3. The tool should help users **check for export restrictions when using FLOSS components**.*

Open source components are developed by communities of different composition and origin. Depending on the specific geographical origin of the open source component, companies using it might have to ensure compliance with certain export restrictions when distributing their products. Ensuring compliance with export restrictions is a complex task and must be automated, which translates into the corresponding tool requirement.

5. Evaluation

This section presents the evaluation of our theory using the feature analysis of existing FLOSS governance tools. We analyzed marketing materials and demos of six widely used FLOSS governance tools. The analysis resulted in the following list of common key features related to FLOSS use in products:

- *Component Tracking & Reporting*: support for a bill of materials, component inventory, knowledge base (external inventory), license obligation reporting, and commonly accepted data exchange standard support
- *Scanning / License Checking*: support for licenses identification, copyright identification, code origin identification, and license management
- *Policies*: support for applying/ensuring FLOSS policies
- *Security*: support for security vulnerability detection
- *Development Integration & Automation*: support for integration into continuous integration and deployment

We focused on two main requirement categories: **Tracking and Reuse of FLOSS components** and **License Compliance of FLOSS**

components. We chose these categories because these requirements are fundamental to any software company according to the analysis of the industry interviews, and tools support of these requirements represent base functionality.

5.1. Tracking and reuse of FLOSS components

The identification of FLOSS components and their licenses in a given software product or component is a core functionality of all sampled tools. All the high-level requirements of category 1 in the proposed theory are matched by the features of the sampled tools. For example, Black Duck Software enables its users to identify the used FLOSS components (Requirement 1.1) in both the source code and in binaries (with lesser precision):

"[Black Duck Hub enables to] fully discover all open source in your code" — Black Duck Hub

FOSSA helps to explore and report relationships between modules incl. the open source ones (Requirement 1.2):

"[FOSSA allows its user to] explore relationships between modules and if/how dependencies are included in your build." — FOSSA

Black Duck Hub also has features for BOM maintenance (Requirement 1.4) and for FLOSS component reuse (Requirement 1.5):

"We provide a license obligation report, including an easily consumable bill of materials (BOM) that you can deliver to your customers and/or internal stakeholders." — Black Duck Hub

"[Black Duck Hub enables to] eliminate uncertainty and promote reuse [of FLOSS]" — Black Duck Hub

However, not all detailed (low-level) requirements from the proposed theory are supported by existing tool features. Requirement 1.1.d, for example, requires tools to allow an automated finding of open source software, not checked-in but used by a company developer. This requirement is not entirely supported by any of the studied tools because of its technological complexity.

5.2. License compliance of FLOSS components

All the studied tools support FLOSS license compliance features. They fulfill requirements, such as license interpretation, license identification, and documentation, FLOSS component approval etc.

FOSSology covers several requirements related to FLOSS license compliance (Requirement 2.2, 2.3) (Gobeille and Robert, 2008):

"FOSSology is an open source license compliance software system and toolkit. As a toolkit, you can run license, copyright and export control scans from the command line. As a system, a database and web UI are provided to give you a compliance workflow. License, copyright, and export scanners are tools available to help with your compliance activities." — FOSSology

Requirement 2.4 (approve FLOSS component use follows guidelines) is covered by WhiteSource. Once policies are created, by using black and white lists of FLOSS licenses, they can be automated applied to a product. Developers can request the approval of a license and the decision for this license will be tracked and archived to make it traceable.

"WhiteSource also lets you create your companys license policy by defining a whitelist of automatically approved licenses; a blacklist of automatically rejected licenses and a list of licenses that need to be approved on a case-by-case basis" — WhiteSource

"These initiate a predefined email approval request, with all approvals tracked, signed and archived within the WhiteSource system for later access." — WhiteSource

The top-level requirement Search and Selection of FLOSS components cant be fulfilled directly by the studied tools, but Black Duck owns and operates the Black Duck Open Hub community platform which fulfills most of the requirements in this category. This platform allows users to search for available FLOSS components, analyze them to select the best one. Open Hub offers analytics about how active the development and community of a component is, and other information that indicates the health of a component.

"[Black Duck Open Hub] is an online community and public directory of free and open source software (FOSS), offering analytics and search services for discovering, evaluating, tracking, and comparing open source code and projects. Where available, the Open Hub also provides information about vulnerabilities and project licenses." — Black Duck Hub

5.3. Architecture model for software products

While the requirements on an architecture model observe FLOSS governance from another point of view they are backing the requirements for FLOSS governance and compliance tools. All of the six categories of requirements on an architecture model can be related to the requirements from the Tables 3–5, and confirm them.

For example, the requirement "Req 4.2. The model should represent **license information"**. about a reusable license model and license policies are related to the requirements of category 2.4 (approve the use of a FLOSS component) and 2.5 (distribute a product that is compliant). A reusable license model allows having a company-based license interpretation that can by applied to all components of a product. It could also help to interpret new licenses by inheriting information from types of licenses, like copyleft licenses, which is also related to the requirements 2.1 (interpret open source licenses). Furthermore, license policies that can be applied to detected licenses are reflected in the requirements of category 2.4.

Another example, are the requirements about automation and the identification of already scanned components. These requirements can be related to all categories of requirements on FLOSS governance tools. For the interviewees it was important that as many tasks as possible could be automated by the tools.

6. Discussion

Our main contribution is the requirements specification presented in Section 4 and its evaluation in Section 5.

Through evaluation, we see that most of the industry requirements are matched by tool providers. However, not all requirements are fulfilled. None of our studied tools completely fulfill some of the following low-level requirements:

- *Requirement 2.1.b* (automated standard interpretation of common FLOSS licenses)
- *Requirement 2.3.h* (automated license checking within continuous integration)
- *Requirement 2.5.b* (automated assignment of FLOSS compliance tasks)
- *Requirement 2.5.c* (automated audit of products bill of materials before distribution)

One reason is the complex computational nature of the complete automation of compliance tasks. An empirical study by German et al. (2010a) showed that a deeper understanding of licensing issues requires human expertise, which limits the automation of some license compliance tasks. Moreover, most companies dont allow complete automation of compliance as they require a human actor to be responsible for legal matters, even if they use semi-automated tooling. Also, the requirements in the category 3.3 (estimate the cost of using a FLOSS component), such as the estimation of costs, risks, and benefits of using FLOSS components cant be fulfilled by any of the studied tools.

Our evaluation demonstrates that the high-level requirements of our theory do match the features offered by industry leading

FLOSS governance tools. The evaluation shows that existing tools satisfy most of the low-level requirements by the industry, but not others, such as requirements of complete automation. We recognize that our research results are limited, but novel and industry relevant. We lay the groundwork for future studies into FLOSS governance tool requirements, that will hopefully expand our requirements specification theory. Our work leads us to propose the following research questions for future research:

- **RQ1**: What are other detailed FLOSS governance tool requirements beyond Tracking and Reuse of FLOSS components, License Compliance of FLOSS components and Search and Selection of FLOSS components?
- **RQ2**: How can FLOSS governance tool requirement theories be better evaluated or validated?
- **RQ3**: How to engineer FLOSS governance tool requirements of the future addressing missing features and industry needs before companies become aware of them?

7. Research limitations

The study faces several limitations. We follow Guba (1981) in assessing the trustworthiness of our research through the quality criteria of credibility, dependability, confirmability, and transferability.

Credibility Credibility is the degree to which we can establish confidence in the truth of our findings in the context of the inquiry. To ensure credibility, we performed two rounds of peer debriefing, together with three colleagues we reviewed this study and incorporated the feedback from our colleagues from within our research group. Furthermore, during data collection we conducted our interviews iteratively, adjusting our semi-structured interview questions based on the company context and on our experience with earlier interviews.

Dependability Dependability is the degree of consistency of the findings and traceability from the data to the results. We ensured dependability by collecting and saving raw interview data, documenting our qualitative data analysis in different stages of the coding and by documenting our analysis in a manner that allows tracing each requirement in our theory to its origin in our collected data. We included numerous direct references to the expert interviews in the presentation of our research findings in Section 4.

Confirmability Confirmability is the degree to which the authors are neutral towards the inquiry and their potential bias effect on the findings. Qualitative data research realized by one researcher has inherent subjectivity and bias. Even though we followed the research method constructs carefully, there is bias associated with method interpretation and application to our specific context. To address this limitation, we had a second coder analyze our data and improve our original QDA coding based on input from the second coder (Lombard et al., 2002).

Transferability Transferability is the degree to which findings of our study hold validity in other contexts. To ensure transferability, we chose companies and tools for our study through a thorough sampling. Though we aimed for a highly representative sample of companies, we do recognize that this study can have a limited degree of transferability as the findings are based solely on the 20 expert interview in eleven companies in our sample. This limitation can be tested through further validation studies.

8. Conclusion

This paper presents a study of eleven industry companies with advanced FLOSS governance practices. Our study concluded in a theory of FLOSS governance tool requirements by the industry. Also, we provide a detailed hierarchical list of these industry relevant requirements. As such it offers unique insight into industry understanding of FLOSS governance tools and their expectations from them, alongside existing tools and their features.

The data gathered through semi-structured interviews and materials collection was analyzed using the novel adoption of grounded theory method: the QDAcity-RE method. We cast our theory as a requirements specification making it applicable and practice relevant to the companies willing to employ these requirements. Finally, we evaluated our findings using six industry-leading FLOSS governance tools and the analysis of their features matched with the requirements of the suggested theory.

The study of the missing features of existing tools is out of the scope of this paper but it can be a valuable part of the further research. Further research can also focus on the reasons why tool providers do not fulfill the unsatisfied requirements of our theory (e.g. full automation of compliance) and how such problems can be solved.

Acknowledgments

This research was funded by BMBFs (Federal Ministry of Education and Research) Software Campus 2.0 project (OSGOV, 01IS17045-17570). We would like to thank Hannes Dohrn, Ann Barcomb, Michael Dorner, Maximilian Capraro, Andreas Kaufmann and Shushanik Hakobyan for their generous feedback that helped us improve our paper. We would also like to thank our industry partners that provided their valuable time and expertise for this research project.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.jss.2019.08.001.

References

Aksulu, A., Wade, M., 2010. A comprehensive review and synthesis of open source research. J. Assoc. Inf. Syst. 11 (11), 576–656 10.1.1.190.4493.

Bonaccorsi, A., Rossi, C., 2003. Why open source software can succeed. Res. Policy 32 (7), 1243–1258. doi:10.1016/S0048-7333(03)00051-9.

Capra, E., Francalanci, C., Merlo, F., 2008. An empirical study on the relationship between software design quality, development effort, and governance in open source projects. IEEE Trans. Softw. Eng. 34 (6), 765–782. doi:10.1109/TSE.2008.68.

Charmaz, K., 2014. Constructing Grounded Theory. Sage.

Corbin, J., Strauss, A., 2014. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage.

Cruz, D., Wieland, T., Ziegler, A., 2006. Evaluation criteria for free/open source software products based on project analysis. Softw. Process Improv. Pract. 11 (2), 107–122. doi:10.1002/spip.257.

De Laat, P.B., 2007. Governance of open source software: state of the art. J. Manage. Governance 11 (2), 165–177. doi:10.1007/s10997-007-9022-9.

Deprez, J.C., Alexandre, S., 2008. Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 5089 LNCS. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 189–203. doi:10.1007/978-3-540-69566-0_17.

Deshpande, A., Riehle, D., 2008. The Total Growth of Open Source, Vol. 275. IFIP International Federation for Information Processing, Springer US, Boston, MA, pp. 197–209. doi:10.1007/978-0-387-09684-1_16.

Di Penta, M., German, D.M., Antoniol, G., 2010. Identifying licensing of jar archives using a code-search approach. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE, pp. 151–160. doi:10.1109/MSR.2010.5463282.

Fitzgerald, 2006. The transformation of open source software. MIS Q. 30 (3), 587. doi:10.2307/25148740.

Franch Gutiérrez, J., Susi, A., Annosi, M.C., Ayala Martínez, C.P., Glott, R., Gross, D., Kenett, R., Mancinelli, F., Ramsany, P., Thomas, C., et al., 2013. Managing risk in open source software adoption. In: Proceedings of the 8th International Joint Conference on Software Technologies (ICSOFT 2013), pp. 258–264.

Gangadharan, G.R., D'Andrea, V., De Paoli, S., Weiss, M., 2012. Managing license compliance in free and open source software development. Inf. Syst. Front. 14 (2), 143–154. doi:10.1007/s10796-009-9180-1.

Gangadharan, G.R., De Paoli, S., D'Andrea, V., Weiss, M., 2008. License compliance issues in free and open source software. In: MCIS 2008 Proceedings, p. 2.

German, D.M., Di Penta, M., Davies, J., 2010. Understanding and auditing the licensing of open source software distributions. In: 2010 IEEE 18th International Conference on Program Comprehension. IEEE, pp. 84–93. doi:10.1109/ICPC.2010.48.

German, D.M., Hassan, A.E., 2009. License integration patterns: addressing license mismatches in component-based development. In: 2009 IEEE 31st International Conference on Software Engineering. IEEE, pp. 188–198. doi:10.1109/ICSE.2009.5070520.

German, D.M., Manabe, Y., Inoue, K., 2010. A sentence-matching method for automatic license identification of source code files. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering - ASE '10. ACM Press, New York, New York, USA, p. 437. doi:10.1145/1858996.1859088.

Gobeille, R., Robert, 2008. The FOSSology project. In: Proceedings of the 2008 International Workshop on Mining Software Repositories - MSR '08. ACM Press, New York, New York, USA, p. 47. doi:10.1145/1370750.1370763.

Gonzalez-Barahona, J.M., Izquierdo-Cortazar, D., Maffulli, S., Robles, G., 2013. Understanding how companies interact with free software communities. IEEE Softw. 30 (5), 38–45.

Guba, E.G., 1981. Criteria for assessing the trustworthiness of naturalistic inquiries. Ectj 29 (2), 75.

Harutyunyan, N., Bauer, A., Riehle, D., 2018. Understanding industry requirements for FLOSS governance tools. In: IFIP International Conference on Open Source Systems. Springer, pp. 151–167.

Hauge, O., Ayala, C., Conradi, R., 2010. Adoption of open source software in software-intensive organizations - A systematic literature review. Inf. Softw. Technol. 52 (11), 1133–1154. doi:10.1016/j.infsof.2010.05.008.

Helmreich, M., Riehle, D., 2011. Best Practices of Adopting Open Source Software in Closed Source Software Products Diplomarbeit im Fach Informatik in Nürnberg. Friedrich-Alexander-Universität ErlangenNürnberg. http://dirkriehle.com/uploads/2011/03/DA-complete.pdf.

Höst, M., Oručević-Alagić, A., 2011. A systematic review of research on open source software in commercial software product development. Inf. Softw. Technol. 53 (6), 616–624.

Hummel, O., Janjic, W., Atkinson, C., 2008. Code conjurer: pulling reusable software out of thin air. IEEE Softw. 25 (5), 45–52. doi:10.1109/MS.2008.110.

Jaeger, T., 2017. Open source license obligations checklists. Open Source Automation Development Lab (self-published white paper), 1–8.

Kapitsaki, G.M., Tselikas, N.D., Foukarakis, I.E., 2015. An insight into license tools for open source software systems. J. Syst. Softw. 102, 72–87.

Kaufmann, A., Riehle, D., 2019. The QDAcity-RE method for structural domain modeling using qualitative data analysis. Requir. Eng. 24 (1), 85–102. doi:10.1007/s00766-017-0284-8.

von Krogh, G., von Hippel, E., 2006. The promise of research on open source software. Manage. Sci. 52 (7), 975–983. doi:10.1287/mnsc.1060.0560.

von Krogh, G., Spaeth, S., Haefliger, S., 2005. Knowledge reuse in open source software: an exploratory study of 15 open source projects. In: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences-Volume 07, 00. 198–2 doi: 10.1109/HICSS.2005.378.

Lakhani, K.R., Von Hippel, E., 2003. How open source software works: "free" user-to-user assistance. Res. Policy 32 (6), 923–943. doi:10.1016/S0048-7333(02)00095-1.

Lattemann, C., Stieglitz, S., 2005. Framework for governance in open source communities. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences. IEEE. 192a–192a doi: 10.1109/HICSS.2005.278.

Lombard, M., Snyder-Duch, J., Bracken, C.C., 2002. Content analysis in mass communication: assessment and reporting of intercoder reliability. Hum. Commun. Res. 28 (4), 587–604. doi:10.1093/hcr/28.4.587.

OpenChain Specification, 2019. https://www.openchainproject.org/spec.

Popp, K.M., 2015. Best Practices for Commercial Use of Open Source Software: Business Models, Processes and Tools for Managing Open Source Software. BoD—Books on Demand.

Radcliffe, M., Odence, P., 2017. The 2017 open source year in review.

Riehle, D., 2007. The economic motivation of open source software: stakeholder perspectives. Computer 40 (4), 25–32. doi:10.1109/MC.2007.147, arXiv:1011.1669v3.

Riehle, D., 2009. The commercial open source business model. In: Lecture Notes in Business Information Processing, Vol. 36 LNBIP, pp. 18–30. doi:10.1007/978-3-642-03132-8_2.

Riehle, D., 2011. Controlling and steering open source projects. Computer 44 (7), 93–96. doi:10.1109/MC.2011.206.

Riehle, D., Harutyunyan, N., 2017. License clearance in software product governance. NII Shonan. http://dirkriehle.com/wp-content/uploads/2017/09/License-Clearance-in-Software-Product-Governance-Public.pdf.

Riehle, D., Lempetzeder, B., 2014. Erfolgsmethoden der Open-Source-Governance und -Compliance. Technical Report. Friedrich-Alexander-Universitat Erlangen-Nürnberg, Erlangen.

Ruffin, M., Ebert, C., 2004. Using open source software in product development: aprimer. doi:10.1109/MS.2004.1259227

Sadowski, B.M., Sadowski-Rasters, G., Duysters, G., 2008. Transition of governance in a mature open software source community: evidence from the Debian case. Inf. Econ. Policy 20 (4), 323–332. doi:10.1016/j.infoecopol.2008.05.001.

Semeteys, R., 2008. Method for Qualification and Selection of Open Source Software. Talent First Network. https://timreview.ca/article/146.

Software, B. D., 2017. 2017 Open Source Security and risk analysis.

Sowe, S.K., Stamelos, I., Angelis, L., 2008. Understanding knowledge sharing activities in free/open source software projects: an empirical study. J. Syst. Softw. 81 (3), 431–446. doi:10.1016/j.jss.2007.03.086.

Stewart, K., Odence, P., Rockett, E., 2012. Software package data exchange (SPDX) specification. Int. Free Open Source Softw. Law Rev. 2 (2), 191–196. doi:10.5033/ifosslr.v4i1.45.

Stol, K.-J., Ali Babar, M., 2010. Challenges in using open source software in product development: a review of the literature. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. ACM, pp. 17–22.

Tools for, 2019. Managing Open Source Programs. https://www.linuxfoundation.org/tools-managing-open-source-programs/.

Umarji, M., Sim, S.E., 2014. Archetypal internet-scale source code searching. In: Finding Source Code on the Web for Remix and Reuse, Vol. 9781461465. Springer US, Boston, MA, pp. 35–52. doi:10.1007/978-1-4614-6596-6_3.

Wang, H., Wang, C., 2001. Open source software adoption: a status report. IEEE Softw. 18 (2), 90–95. doi:10.1109/52.914753.

Nikolay Harutyunyan, M.Sc., is a researcher and Ph.D. student at the Professorship for Open Source Software at Friedrich-Alexander University Erlangen-Nürnberg (FAU). Nikolays research focuses on corporate open source governance and user experience design. Before joining the research group of the Professorship for Open Source Software, he studied Information Systems at FAU.

Andreas Bauer, M.Sc., is a researcher and Ph.D. student at the Professorship for Open Source Software at Friedrich-Alexander University Erlangen-Nürnberg (FAU). Andreas research focuses on license compliance tooling and open source software development. Before joining the research group of the Professorship for Open Source Software, he studied Computer Science at FAU.

Dirk Riehle, M.B.A., is the Professor of Open Source Software at the Friedrich-Alexander University Erlangen-Nürnberg. Before joining academia, Riehle led the Open Source Research Group at SAP Labs, LLC, in Palo Alto, California (Silicon Valley). Riehle founded the Open Symposium (OpenSym, formerly WikiSym). He was the lead architect of the first UML virtual machine. He works on open source and inner source software engineering as well as agile software development methods and continuous delivery. Prof. Riehle holds a Ph.D. in computer science from ETH Zürich and an M.B.A. from Stanford Graduate School of Business.