Industry Best Practices for Component Approval in FLOSS Governance

Nikolay Harutyunyan
Computer Science Department
Friedrich-Alexander University Erlangen Nürnberg
Erlangen, Germany
nikolay.harutyunyan@fau.de

Dirk Riehle
Computer Science Department
Friedrich-Alexander University Erlangen Nürnberg
Erlangen, Germany
dirk@riehle.org

ABSTRACT

Increasingly companies realize the value of using free/libre and open source software (FLOSS) in their products, but need to manage the associated risks. Leading companies introduce open source governance as a solution. A key aspect of corporate FLOSS governance deals with choosing and evaluating open source components for use in products. Following an industry-based research approach, we present 13 best practices in the pattern format of context-problem-solutions paired with consequences. In this paper, we cover an excerpt of the Component Approval section of our FLOSS governance handbook. This article builds upon our previous EuroPLoP publication covering Component Reuse in FLOSS governance processes, as well as other publications on the topic. Analyzing qualitative data gathered from 15 expert interviews, we derive and interconnect the common industry recommendations for reviewing, tracking, and approving open source components in a company environment. We conclude by presenting workflow templates that put various best practices in relation to each other.

KEYWORDS

Best Practice, Commercial Use of Open Source, Component Approval, FLOSS, FOSS, Industry Best Practice, Open Source Software, Open Source Governance, Pattern, Pattern Language

ACM Reference format:

Nikolay Harutyunyan and Dirk Riehle. 2020. Industry Best Practices for Component Approval in FLOSS Governance. In *Proceedings of EuroPLoP 2020. ACM, New York, NY, USA, 12 pages.* https://doi.org/10.1145/3424771.3424791

1 Introduction

Using open source software components in products carries potential risks for companies including legal, business, and technical risks resulting from mishandling of open source licenses, export restrictions, copyright notices, and software supply chains [21, 22, 24]. While some companies have realized these risks and establish internal guidelines and rules for open source governance, many companies remain either ignorant or unaware of such threats, while using FLOSS components as part of their commercial products.

As academic literature on the topic of open source governance is lacking, we studied practitioner articles and reports to motivate our industry-sourced study. In the context of this study, we define FLOSS governance as the set of processes, best practices, and tools employed by companies to use FLOSS components as parts of their commercial products while minimizing their risks and maximizing their benefit from such use [11, 12]. We only focus on the governance aspects of companies using open source software, not their contributions to or leadership of open source communities. We covered some of the latter in a recent publication [13], where we also used the pattern format of context-problem-solutions and plan to continue our work in that direction.

Given the different maturity levels of companies when it comes to open source governance, different aspects have higher priority. After getting started with open source governance, companies should define processes for choosing and reviewing potential FLOSS components and libraries that are to be checked into company code repositories. Being a crucial topic for all companies, we asked the following research question:

Research question: How should companies approve the proposed use of open source components in the context of open source governance based on existent industry best practices?

To answer the research question, we used the qualitative survey method [17]. We selected 15 companies from a pool of 140 with expert knowledge on the topic of open source governance and interviewed their experts including software developers, technical and business managers, as well as legal counsels. Analyzing the gathered data, we found common themes and recommendations on Component Approval in the context of FLOSS governance. We aimed to have a polar sample with diverse companies from small consultancies to large automotive companies. We do not claim a representative sample, but rather present exploratory findings from a diverse mix of companies with state-of-the-art best practices for open source governance. We contribute to the academic community by proposing a theory of open source governance focused on Component Approval. We also contribute to the patterns community by casting our findings as 13 context-problem-solution patterns coupled with consequences and building upon our previous work on Component Reuse as part of FLOSS governance [15].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
EuroPLoP '20, July 1–4, 2020, Virtual Event, Germany
© 2020 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7769-0/20/07…$15.00
https://doi.org/10.1145/3424771.3424791

While both papers use the data from our larger study into open source governance, in this paper we present novel findings on a previously unstudied topic of open source component approval in companies. This is a topical predecessor to our 2019 paper on component reuse, where we presented best practices for reusing open source software already inside company products. These are distinct and equally important topics of the inbound FLOSS governance, alongside software supply chain management, which is another topic we plan to address in future publications.

Some of the best practices in this paper suggest the creation and maintenance of an open source component repository. Other practices help define transparent rules for open source component approval. The proposed best practices come together in workflows or process templates that any interested practitioner can employ and adapt to their use case. Aiming for a high level of industry relevance, we thus tried to increase the applicability of our results, while not compromising the research rigor.

In the context of this paper, we use patterns to represent best practices. The best practices we identified are procedural and fit together into systematic processes. Thus, we also use patterns to capture the sequential (and at times the hierarchical) nature of the best practices. As a result, we use the terms best practices, patterns, and workflow components interchangeably taking into account the important distinctions mentioned above.

2 Related Work

Conceptually we differentiate between inbound and outbound open source governance. The former focuses on managing how open source components get into the company, while the latter covers managing the distribution of products that include open source components. As component approval is part of the inbound governance, we studied the limited research available on the topic, including engineering management and software development [2, 7, 16, 24], open source component search [8, 20], open source component selection [1, 4], open source component approval [9, 18], and open source component reuse [3, 8].

In our own previous work, we already covered some of the related open source governance topics, such as Getting Started with FLOSS governance [14], Component Reuse [15], and Tooling [11, 12]. Building upon the FLOSS governance patterns from these publications [14, 15], this paper adds industry best practice patterns for open source Component Approval.

In the context of FLOSS component approval in companies, Glynn et al. [9] discussed the commercial adoption of open source software through an empirical study followed by a survey. Adapting the OSS Assimilation Levels framework by Fichman and Kemerer [5], they find that companies with a basic open source governance awareness aimed to evaluate the open source components they were about to use. In our study, we confirm that companies focus on open source component review and evaluation following the establishment of the initial governance infrastructure during a getting started process. This is reflected in the best practice patterns *OSGOV-COMAPP-4.* and *OSGOV-COMAPP-1.*

Koltun [18] wrote about the commercial use of open source software and license compliance as part of FLOSS governance. One of his key insights addresses the review and approval of the open source components used in company products. Namely, he finds that an open source review board (OSRB) in a company should review FLOSS use in the context of the use case and license of a given component. We find similar recommendations based on our expert interview analysis, captured in the pattern *OSGOV-COMAPP-11.* where we cover analyzing code for license compliance, and in *OSGOV-COMAPP-12.* on reviewing FLOSS use in the context of product architecture.

As to the presentation of our findings, we chose patterns following successful examples and learning from related research [19, 23, 25, 26]. Following up on our previous work we aim to develop an all-encompassing pattern language for commercial open source governance in the future. This publication constitutes another building block in this effort.

3 Related Method

We asked the research question on how companies should review, track, and approve the proposed use of open source components in the context of open source governance. To answer this question, we conducted a qualitative survey using interviews with industry experts to collect data [6, 17]. We collected background information, designed and planned the study, conducted several rounds of sampling, and chose 15 experts to interview (employees involved with designing and implementing open source governance were interviewed). We defined interview questions that covered different aspects of open source governance and component approval among other topics. As this was part of a larger study, we used the same interviews for our study on a related topic of open source governance and component reuse which resulted in a number of best practices on that topic [15].

The resulting semi-structured interviews were conducted in an iterative manner. We then analyzed the survey data employing qualitative data analysis (QDA) aided by a QDA tool called MAXQDA to ensure the systematic analysis and traceability to our findings. Finally, we presented our theory reflecting the state-of-the-art best practices for commercial open FLOSS governance and component approval.

As a result of an iterative sampling process from our network of about 140 organizations with advanced FLOSS governance awareness, we chose 14 companies and one open source foundation. To ensure a diverse set of sources we classified the companies and foundations in our professional network by business domain, size (based on the revenue and number of employees), type of customers, business models, etc. The list of companies and some of their characteristics are presented in Table 1. Company names are anonymized per their request.

Table 1. Theoretical sample of companies

Company	Company domain	By size	By type of customer
Company 1	Consulting	Medium	Enterprise
Company 2	Automotive	Small	Enterprise
Company 3	Automotive	Large	Enterprise
Company 4	Enterprise Software	Medium	Enterprise, retail
Company 5	Enterprise Software	Medium	Enterprise, retail

Company 6	Enterprise Software	Large	Enterprise, retail
Company 7	Enterprise Software	Medium	Enterprise, retail
Company 8	FLOSS Foundation	Small	Enterprise, retail
Company 9	Hardware and Software	Large	Enterprise
Company 10	Legal	Large	Enterprise, government
Company 11	Enterprise Software	Medium	Enterprise
Company 12	Consulting, Enterprise Software	Large	Enterprise
Company 13	Hardware and Software	Large	Enterprise, retail, government
Company 14	Enterprise Software	Small	Enterprise
Company 15	Enterprise Software	Large	Enterprise

4 Research Results

Answering our research question, we propose a set of industry best practices for open source governance in the context of FLOSS component approval in companies. Through our qualitative survey, we find 13 patterns common across 15 companies we studied. We present each best practice using context-problem-solution patterns that are interconnected forming a section in an open source governance handbook on component approval.

Our patterns are linked forming process templates that can be used by practitioners looking to apply our theory in their companies. At the end of this section, we present some examples of such processes. Not all best practices need to be applied in order to achieve the company's goals. Our findings are abstract recommendations based on companies with different contexts. Therefore, to apply our theory in practice, one should invest the time to adjust and detail the proposed solutions to the specific application contexts.

Our theory of industry best practices for open source governance and component approval includes:

- definition of component approval process
- operationalization of component approval process
- definition of component approval rules
- design and review of component approval requests
- guidelines for making, communicating, and appealing component review decisions
- component analysis in the context of license compliance and product architecture.

Our theory focuses on the early maturity levels of open source governance in companies. While companies need to start by establishing a successful transition from ungoverned use of open source first, they then need to follow up by designing and operationalizing a review process for the FLOSS components developers want to check into the company codebase. We find that a key actor responsible for open source component approval in companies is the Open Source Program Office (OSPO). While the title changes from one company to the other, OSPO's responsibilities and tasks are consistent, even though different in scale depending on the company. Generally, we find that practitioners (and OSPOs in particular) should start by defining transparent rules for component review, alongside a component evaluation process that should take as input key component data, including among others:

- component name
- component address / location
- product / project name
- open source license name
- multiple licenses (y/n)
- open source license version
- copyright holder
- linkage type to the rest of the (software) product (e.g. dynamic or static)
- use case (e.g. internal use only, to be directly distributed as part of the product, used to compile software to be distributed as part of the product).

Covering the above-mentioned aspects of open source governance and beyond, we present the proposed theory in its entirety as a collection of interconnected patterns (presented by "→" within the patterns). The overview of these patterns is presented as follows:

OSGOV-COMAPP-1. Define the component approval process
OSGOV-COMAPP-2. File a component approval request
OSGOV-COMAPP-3. Review a component approval request
OSGOV-COMAPP-4. Define transparent rules for open source component approval
OSGOV-COMAPP-5. Communicate open source component approval rules
OSGOV-COMAPP-6. Make a component approval decision
OSGOV-COMAPP-7. Appeal a component approval decision
OSGOV-COMAPP-8. Communicate component approval process
OSGOV-COMAPP-9. Implement component approval process
OSGOV-COMAPP-10. Provide approval request templates
OSGOV-COMAPP-11. Analyze code for license compliance
OSGOV-COMAPP-12. Review use in the context of product architecture
OSGOV-COMAPP-13. Add decision to component repository.

The above-mentioned best practices are presented in full as follows.

OSGOV-COMAPP-1. Define the component approval process

Name	Define the component approval process
Actor	OSPO (Open Source Program Office)
Context	One of the key aspects of open source governance is component approval. Software developers routinely go through a process of searching, selecting, approving, and integrating software components into the company's products.
Problem	Using open source components has its unique complexities, such as considering open source

	licenses, their obligations, and interdependencies. Without a systematic approach, companies end up having no or inconsistent open source governance in place depending on the awareness of a given employee or team. How can a company systematically check and approve the use of open source components?
Solution	Companies using open source components in their products need to establish components approval processes that follow → *component search* and → *component selection*. OSPO must define a streamlined component approval process that does not hinder the production, but reliably ensures that the selected open source component can be used in the product without any negative side effects. The component approval process consists of: - *filing a request* - *reviewing a request* - *making a decision* - *appealing a decision.* The component approval process can be assisted by tools as part of the production toolchain, in order to automate the request and decision submission, and communication.
Conse-quences	By defining the component approval process, the OSPO: + restricts the otherwise ad-hoc approach for open source component review, thus limiting the risks of ungoverned FLOSS use in company products + prescribes concrete roles and responsibilities related to open source component approval at a high level - does not provide low level guidance on how to implement the steps of open source component review.

OSGOV-COMAPP-2. File a component approval request

Name	File a component approval request
Actor	Developers
Context	Software developers → *select components* on their own based on their functional and non-functional requirements. However, before the usage, open source components need to be checked and approved by the OSPO or by following → *OSPO's predefined rules*.
Problem	How can software developers inform the OSPO about their intention to use a certain open source component?
Solution	The developer must fill in an OSPO-provided checklist for an open source component. The checklist includes the following data about the components: - Component ID - Component name - Component address / location - Product / Project ID - Product version - Product / Project name - Open source license name - Multiple licenses (y/n) - Open source license version - Copyright holder - Linkage type to the rest of the (software) product (e.g. dynamic or static) - Use case (e.g. internal use only, to be directly distributed as part of the product, used to compile software to be distributed as part of the product) - Has the component (with its unchanged license and version) been used in the company before (can be automatically identified)? OSPO can define and communicate additional points in the checklist if necessary. The developer must file the request with the complete checklist to the OSPO that → *reviews the request*. Filing a request can be done using → *provided approval request templates*, or it can be automated to an extent and integrated into the development toolchain to ensure efficiency and ease of use.
Conse-quences	By filing component approval requests, software developers: + increase their certainty on conforming with company rules for open source use, thus reducing the risks of open source license non-compliance + increase development efficiency by setting up and following a standard procedure for the common and frequent task of vetting an open source component to be used - create an overhead to their core responsibility for product development, which can decrease the speed of development and result in longer timelines.

OSGOV-COMAPP-3. Review a component approval request

Name	Review a component approval request
Actor	OSPO (Open Source Program Office)
Context	Software developers → *file component approval requests* to OSPO. OSPO needs to respond to these requests as soon as possible.
Problem	How can OSPO review open source component approval requests in an efficient and timely manner?

Solution	OSPO receives a checklist with each developer-filled component approval request. OSPO needs to review these requests in an efficient and timely manner in order not to hinder the production by becoming a bottleneck. To ensure this OSPO needs to → *define transparent rules for open source component approval* and → *communicate the rules for open source component approval*. These rules help make the → *decision making process* easier, because some licenses-use case pairs can be automatically approved, while some other pairs can be automatically rejected. Moreover, the review of approval claims takes into account whether the given component with this license and license version has been used in the company (in the same use case as requested) in the past. If that's the case, the use of the component is automatically reviewed and approved (or rejected if such a decision was recorded). As a result, the OSPO only reviews the new license/use case pairs of open source components, after which a decision is made and → *the decision is documented* for future reference. To review a new license/use case pair, the OSPO assesses the technical, legal, and business implications of the open source component use. If these correspond to the company's policy towards open source governance, OSPO → *analyzes code for license compliance*, which can be assisted by open source compliance scanners or other tools. You can → *use open source governance tools to make the open source component approval more efficient*, helping to automate the decisions that can be resolved without the involvement of the OSPO.
Conse-quences	By filing component approval requests, software developers: + increases the transparency and efficiency of FLOSS component review independent of the developer or team filing the request + streamlines the review process and integrates it into the existing production workflow - reduces the flexibility and speed of the review that was previously based on personal relationships and (FLOSS-related) experience of the developer filing the request.

OSGOV-COMAPP-4. Define transparent rules for open source component approval

Name	Define transparent rules for open source component approval
Actor	OSPO (Open Source Program Office)
Context	Software developers → file component approval requests to OSPO. OSPO → reviews component approval requests. For this review, OSPO must define consistent, transparent and traceable rules.
Problem	How can OSPO define rules for open source component approval review?
Solution	OSPO must define and communicate transparent rules for open source component approval. These rules are based on open source license(s) of the component and its intended use case in the final product. Having such rules enables developers to take open source licenses and use cases into consideration during → *component search* and → *component selection*. OSPO must define: • open source licenses that contradict the company's open source governance policy for any use case • open source licenses/use case pairs that contradict company's open source governance policy • open source licenses/use case pairs that correspond to the company's open source governance policy. For these three situations, OSPO must define component approval rules that can be automated using open source governance tools. If a developer → *files a component approval request* with a checklist that matches one of the above-mentioned scenarios, the rules should automatically approve or reject the open source component request. This applies only to the known or recorded → *license/use case pairs that OSPO inherits from the transition board* (during the getting started process) that are based on the → standard license interpretations developed during the getting started process. The rules can be modified and adjusted by OSPO as needed.
Conse-quences	By defining transparent rules for open source component approval, the OSPO: + establishes certain, centralized, and standardized approach to dealing with open source components + educates the developers on the accepted use of FLOSS components in different use cases - could discourage developers from experimenting with open source software that can be deemed non-conformant to the rules, even though technologically superior.

OSGOV-COMAPP-5. Communicate open source component approval rules

Name	Communicate open source component approval rules
Actor	OSPO (Open Source Program Office)
Context	After → defining transparent rules for open source component approval, it is necessary to make the rules accessible to the employees, so they can follow them during → *component search* and → *component selection*.
Problem	What are the best channels to communicate the open source component approval rules?
Solution	Communicate open source component approval rules using a variety of channels: ● transition board's → *communication channels* for open source governance handbook ● multi-purpose internal communication channels, such as intranet, wikis, forums, etc. ● open source governance tools integrated into development toolchain. It is also recommended to communicate the details of the policy through employee trainings that include other topics around open source governance.
Conse-quences	By communicating open source component approval rules, the OSPO: + explicitly announces the newly established open source component review rules for software developers, decreasing the unnecessary complexity of the informal governance and sending the message on a new structured approach for governance - creates additional overhead for software developers as they need to familiarize themselves with the new rules and may need additional training.

OSGOV-COMAPP-6. Make a component approval decision

Name	Make a component approval decision
Actor	OSPO (Open Source Program Office)
Context	Software developers → *file component approval requests* to OSPO. OSPO → *reviews component approval requests*. Now OSPO needs to make a decision whether to approve or reject the use of the given open source component in the product.
Problem	How should OSPO make a decision about component approval requests?
Solution	OSPO must first double check if the component can be automatically approved or rejected. This applies only to the previously used license/use case pairs, meaning the requested open source license has already been used in the requested use case. OSPO refers to its → *defined rules for open source component approval* and its previous → *decisions added to component repository*. The following decisions are taken: • if open source licenses contradict the company's open source governance policy for all use cases, then the component is automatically rejected • if open source licenses/use case pairs contradict the company's open source governance policy, then the component is automatically rejected • if open source licenses/use case pairs correspond to the company's open source governance policy, then the component is automatically approved. For situations where the open source license and/or the use case are new to the company, OSPO needs to → *analyze code for license compliance*, while assessing its use case. After this OSPO (supported by the legal department) must decide if the new license/use case pair corresponds to the company's open source governance policy. To decide OSPO hears the assessment of its legal and business decision-maker members. OSPO also → *reviews open source component use in context of product architecture*. Once an approval or rejection decision has been made, OSPO → *adds this decision to component repository*. The developer who submitted the component approval request can → *appeal a component approval decision* to the Open Source Program Officer.
Conse-quences	By making component approval decisions, the OSPO: + employs open source component review automation when possible, which leads to reduced overhead for software developers and other stakeholders + follows the pre-defined rules for the manual review cases, thus rendering the decision making more objective and acceptable to the requesting party - adds considerable overhead for the case-by-case review of complex requests.

OSGOV-COMAPP-7. Appeal a component approval decision

Name	Make a component approval decision
Actor	Developer
Context	OSPO → *reviews component approval requests* and → *makes approval or rejection decisions*. However, OSPO can make mistakes, so developers need a channel to appeal component rejection decisions.
Problem	How can a developer appeal OSPO's component rejection decision?
Solution	Developers can appeal OSPO's component rejection decision to the Open Source Program Officer. This appeal must at least include: - open source component information - license/use case assessment by the developer - argumentation for the appeal. Open Source Program Officer reviews the appeal, consults with other members of the OSPO, legal, and business management if needed. The Open Source Program Officer then makes a final decision and sends it to the developer. If the decision is different from OSPO's original decision, OSPO → *adds this decision to component repository*.
Conse-quences	By making component approval decisions, the OSPO: + deals with the outlier or special cases and ensures that the decision making remains aligned with the production goals and requirements, even when the open source component review is controversial + adds the necessary flexibility to the open source component review process and ensures that the semi-automated decision-making process is not merely a formality - can create conflicts or unnecessary escalations with software developers with a strong inclination of using an open source component that is ultimately rejected.

OSGOV-COMAPP-8. Communicate component approval process

Name	Communicate component approval process
Actor	OSPO (Open Source Program Office)
Context	After → *defining the component approval process*, it is necessary to make the process accessible to the employees, so they can follow it.
Problem	How should you store and share the used open source components, their metadata and reuse information across the organization?
Solution	Communicate open source component approval process using a variety of channels: • transition board's → *communication channels for open source governance handbook* • multi-purpose internal communication channels, such as intranet, wikis, forums, etc. • open source governance tools integrated into development toolchain. It is also recommended to communicate the details of the policy through → *employee training that includes strategic topics around open source governance* and → *employee training that includes specialized topics around open source governance*.
Conse-quences	By communicating the component approval process (similar to the pattern on communicating the component approval rules), the OSPO: + explicitly announces the newly established open source component approval process replacing the previously informal governance - creates additional overhead for software developers and other stakeholders of the process, which can require additional education and coordination.

OSGOV-COMAPP-9. Implement component approval process

Name	Implement component approval process
Actor	OSPO (Open Source Program Office)
Context	After → *defining component approval process* and → *communicating component approval process*, OSPO needs to implement the component approval process and transition towards this institutionalized approach.
Problem	How can OSPO implement a component approval process?
Solution	You implement the established process that covers the essentials of the component approval process. OSPO implements the process gradually, first introducing the overall process to the affected team, then demonstrating an example of filing and reviewing an approval request. The decision-making rules are explained to the affected developers. Along the way, Q&A sessions and discussions between developers and OSPO ensure a smooth implementation and process evaluation. The end goal of the process implementation is to ensure that developers understand the changes introduced by this handbook, as well as the motivation behind these changes.

Conse-quences	By implementing the component approval process, the OSPO: + operationalizes the newly defined process and integrates it into the existing workflows of the company, which results in an efficient application of the process while minimizing the disruptions to the production - can result in unexpected disruptions to software development if done carelessly and without the engagement of the affected stakeholders.

OSGOV-COMAPP-10. Provide approval request templates

Name	Provide approval request templates
Actor	OSPO (Open Source Program Office)
Context	After → *defining component approval process* and → *communicating component approval process*, developers start using the process. The first step is for developers to → *file component approval requests*.
Problem	How can OSPO make it easier for developers to file component approval requests?
Solution	To make it easier for developers to → *file component approval requests* OSPO can: • create and provide approval request templates • integrate approval request templates in tools as part of overall development toolchain. The template needs to have only the essential information of the request, including: • name and ID of the developer • name and ID of the organizational unit • component approval checklist needed to → *file a component approval request*: o Component ID o Component name o Component address / location o Product / Project ID o Product version o Product / Project name o Open source license name o Multiple licenses (y/n) o Open source license version o Copyright holder o Linkage type to the rest of the (software) product (e.g. dynamic or static) o Use case (e.g. internal use only, to be directly distributed as part of the product, used to compile software to be distributed as part of the product) o Has the component (with its unchanged license and version) been used in the company before (can be automatically identified) This template needs to be automated and pre-filled to the extent that it is possible using tool integration and open source governance tools (e.g. license scanners, software component management tools, etc.).
Conse-quences	By providing approval request templates, the OSPO: + creates a standardized and easy-to-follow way of conforming with the newly established open source component approval process, which saves employee time and overhead + ensures that the newly established open source component approval process is followed even if some employees did not receive the training on the process - can end up receiving incomplete data and/or metadata on the given open source component, if the template does not explicitly request it.

OSGOV-COMAPP-11. Analyze code for license compliance

Name	Analyze code for license compliance
Actor	OSPO (Open Source Program Office)
Context	Software developers → *file component approval requests* to OSPO. OSPO → *reviews*

	component approval requests. During the review, you need to assess license compliance.
Problem	How can OSPO analyze the requested code for license compliance?
Solution	OSPO needs to look at the open source licenses and the use case of the component from the filed component approval requested in the component approval request. It also needs to look at the open source licenses declared in the checklist by the developer in the submitted component approval requested. OSPO then needs to analyze the source code of the open source component to verify if all the open source licenses are correctly identified in the checklist. This includes checking the precise license text and whether it has been modified. This affects license compliance and the resulting obligations for the company. If all the licenses are correctly identified, OSPO needs to assess the code for license compliance with company policy using its → *defined rules for open source component approval*. If other open source licenses are identified in the component, each needs to be assessed for license compliance using the → *defined rules for open source component approval* or individual case by case analysis for the new license/use case pairs. Moreover, OSPO needs to assess the license compliance of license mixtures to avoid incompatible open source licenses and communicate to the developer what problems were found in the application, so that the developer learns how to check more effectively. When analyzing code for license compliance, OSPO should employ open source code/license scanners and other open source governance tools to ensure efficiency.
Conse-quences	By analyzing code for license compliance, the OSPO: + aligns the governance process for open source license checking and that for open source component approval, which results in a more coordinated and efficient approach to using open source in products - lengthens the duration of the review due to the need for additional license scanning, even though the component might not be ultimately used in products.

OSGOV-COMAPP-12. Review use in the context of product architecture

Name	Review use in the context of product architecture
Actor	OSPO (Open Source Program Office)
Context	Software developers → *file component approval requests* to OSPO. OSPO → *reviews component approval requests*. During the review, you need to assess how the open source component fits into the product architecture.
Problem	Open source components will become part of the product architecture. Different interdependencies that are created can affect the component approval process because of the interactions between licenses. How should OSPO review the use of open source components as part of product architecture?
Solution	OSPO needs to assess the technical and legal effects of adding an open source component to the product architecture. Technical issues include whether the open source component can be easily integrated once approved. Legal issues include whether the integration of the open source component under its current license is not problematic now or will not be problematic in the future (e.g. in terms of open source license compatibility or mixing licenses). OSPO inherited the initial product architecture from the getting started process after → *running open source use analysis in products* and → *documenting current open source use*. OPSO can use open source governance tools to simulate the effect of adding the requested component into the current product architecture. The results of this review feed into → *reviewing a component approval request*.
Conse-quences	By reviewing the open source component use in the context of product architecture, the OSPO: + aligns the software architecture (software components, their data, and interconnections) with the governance process for open source component approval, which results in unified storage of the FLOSS-specific metadata in the overall product architecture, thus making its documentation more efficient - can introduce additional complexity to the overall product architecture, that might not be relevant to all the employees using the product architecture and thus negatively impact its readability.

OSGOV-COMAPP-13. Add decision to component repository

Name	Add decision to component repository
Actor	OSPO (Open Source Program Office)
Context	OSPO → *reviews component approval requests* and → *makes approval or rejection decisions*. Developers can → *appeal a component approval decision*. Once that's settled, the final

	approval or rejection decision must be documented for future use and reference.
Problem	How should OSPO document its component approval or rejection decisions?
Solution	OSPO needs to set up a component repository based on the → *documented current open source use* from the getting started process. This component repository includes the relevant information about all open source component approval requests, including the data from the filed request checklist. The repository includes information on all the requests with both approval and rejection decisions by the OSPO. Each decision can have an optional memo to explain the decision to the employees. The repository should be open to the developers of the company, so they can consult it before making a new request, as the same license/use case pair may have already been assessed in the past. In the latter situation, the developer still needs to make a formal request, however, it will be automatically approved if the same license/use case pairs have already been approved in the past. The repository needs to be searchable and easy to use for developers and for OSPO. It should have well-defined structure and be maintained by OSPO, preferably supported by an open source governance tool and using a common (standard) format for such data (e.g. SPDX). The latter can be useful for supply chain management and outbound governance.
Conse-quences	By adding decisions to the component repository, the OSPO: + documents the log of open source component use requests, approvals, rejections, and the related metadata and reasoning in a semi-automated and consistent manner that can save time in the future if another developer (from any team at the company) makes a similar request - needs to maintain the component repository and ensure its consistency over time, as employees might make incomplete or redundant requests.

5 Discussion and Conclusion

This study follows our previous work on the topic of inbound open source governance focused on component reuse. Leveraging the rich data from our qualitative survey, we propose a theory of industry best practices of FLOSS governance and component approval in this paper.

When it comes to FLOSS component review, tracking, and approval, we found that companies should start by defining a process that guides developers, managers, and other stakeholders in selecting and evaluating the components to be used as part of a product. Following the process definition for component approval, companies should operationalize it by setting up company guidelines and workflows that encompass how approval requests are made, review decisions are determined and documented. Our best practices address these issues in detail.

Known uses are an important part of a pattern. We trace the known uses of our proposed best practices to our data of expert interviews, as well as other supporting literature both academic and practitioner. Let's discuss some examples of such known uses below. The first best practice *OSGOV-COMAPP-1. Define the component approval process* is an overview of the common approach all companies had towards the issue of systematic open source component approval. For example, an expert from Company 2 (a small automotive supplier company) talks about their process for component selection and approval. Given the small size of their company, their process is not formalized using checklists (common approach for the company), but rather is part of the documentation process that employees have to follow when using open source software in production. Another known use for the same pattern was observed at Company 6 (a large enterprise software vendor), where a whole team of 4 full-time employees was working on open source component review and approval. Given the much larger scale of the company and its critical reliance on open source software in products, the open source review team had a formal and clearly defined process for component approval. As suggested in the captured best practices, this process included filing approval requests by software developers, reviewing such requests and making approval decisions by the team after checking for open source license compliance and the fit of the component into the overarching product architecture. As yet another known use of the pattern, the interviewed open source governance expert from Company 9 (a large producer of both hardware and software) presented their component approval process, which was systematically integrated with the outbound governance process. When checking an open source component for use in products, the process would make sure that the details of the component approval were documented and checked in a manner that allowed for a quicker review of the final product before the release (outbound review). Beyond the known uses in the companies we studied directly, we also found other known uses of the same best practices reported in the literature. In particular, practitioner literature such as corporate guidelines often confirmed our findings. Continuing the example of the *OSGOV-COMAPP-1.* pattern, we found that it was confirmed by leading FLOSS governance experts such as Jeff McAffer (Director of Open Source Program Office) of Microsoft and Ibrahim Haddad (VP of R&D, Head of Open Source) of Samsung Research America. As for academic literature, we found fewer known uses of our best practices there, which was our initial expectation and reason for taking a practice-oriented approach that would help capture the state-of-the-art practices in the industry yet unreported by researchers. However, an example of a known use from the academic literature is Koltun's paper on FLOSS compliance [8], which provided some known uses for our best practices *OSGOV-COMAPP-6. Make a component approval decision*, *OSGOV-COMAPP-11. Analyze code for license compliance*, and *OSGOV-COMAPP-12. Review use in the context of product*

architecture. Namely, Koltun described how an Open Source Review Board or OSRB of a company should make approval recommendations for given open source components, as well as guidance on checking open source license compliance and reviewing how such a component would fit the overall product architecture.

Another important property of the patterns we propose is their interconnection both hierarchical and sequential. Using the latter, we defined exemplary workflows consisting of some of the discovered best practices. Such workflows can enable practitioners to apply our findings in an industry context with some adjustment and specification. Figure 1 and Figure 2 illustrate two examples of process templates for open source component approval.

6 Research Limitations

We recognize that our study has some limitations and follow Guba [10] in assessing the trustworthiness of our research through the quality criteria of credibility and dependability.

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 during data collection we conducted our interviews iteratively, adjusting our semi-structured interview questions based on the company's context and on our experience with earlier interviews.

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 best practice in our theory to its origin in the data.

ACKNOWLEDGMENTS

This research was funded by BMBF's (Federal Ministry of Education and Research) Software Campus 2.0 project (OSGOV, 01IS17045-17570). We would like to thank our colleagues and the anonymous reviewers for their feedback. We would also like to thank our industry partners that provided their valuable time and expertise for this research project.

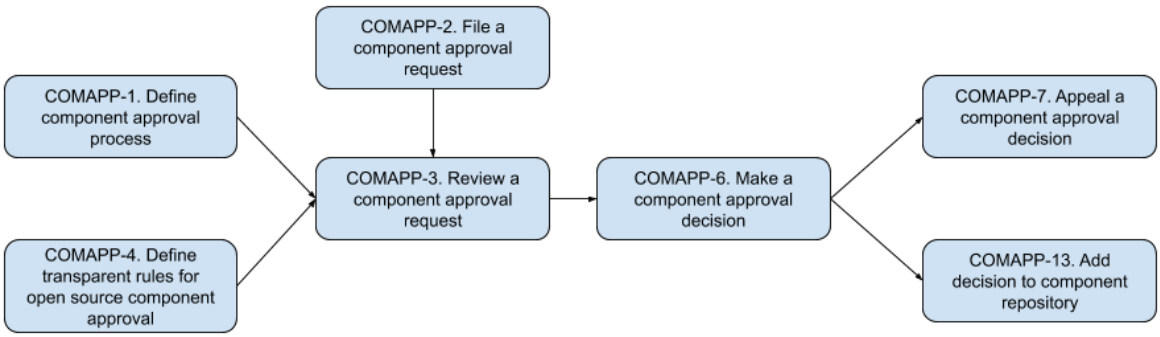

Figure 1. An example of a workflow diagram for FLOSS Governance and Component Approval - Process Template A

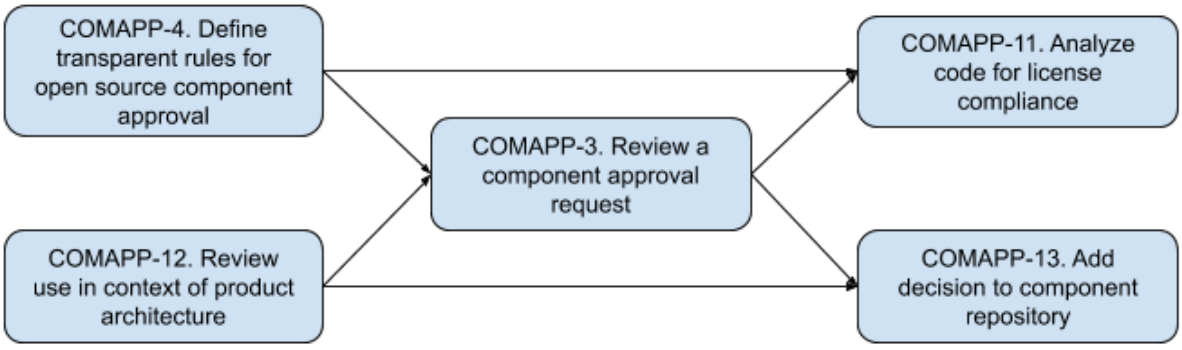

Figure 2. Another example of a workflow diagram for FLOSS Governance and Component Approval - Process Template B

REFERENCES

[1] Ardagna, C. A., Banzi, M., Damiani, E., & Frati, F.: Implementing open source software governance in real software assurance processes. In International Conference of Software Business. Springer, 103–114 (2010)

[2] Berglund, E., Priestley, M.: Open-source documentation: in search of user-driven, just-in-time writing. In Proceedings of the 19th Annual International Conference on Computer Documentation. ACM, 132–141 (2001)

[3] Brown, A. W., Booch, G.: Reusing open source software and practices: The impact of open-source on commercial vendors. In International Conference on Software Reuse. Springer, 123-136 (2002)

[4] Fendt, O., Jaeger, M., & Serrano, R. J.: Industrial experience with open source software process management. In 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), volume 2. IEEE, 180–185 (2016)

[5] Fichman, R. G., Kemerer, C. F.: The Assimilation of Software Process Innovations: An Organizational Learning Perspective, Management Science (43:10), 1345-1363 (1997)

[6] Fink, A.: Analysis of qualitative surveys. In: The survey handbook, 61–78. SAGE Publications, California (2003)
[7] Fitzgerald, B.: The transformation of open source software. MIS Quarterly, 587–598 (2006)
[8] German, D. & Di Penta, M.: A method for open source license compliance of java applications. IEEE Software, 29(3), 58–63 (2012)
[9] Glynn, E., Fitzgerald, B., & Exton, C.: Commercial adoption of open source software: an empirical study. In 2005 International Symposium on Empirical Software Engineering: IEEE (2005)
[10] Guba, E. G.: Criteria for assessing the trustworthiness of naturalistic inquiries. In: Educational Technology Research and Development, 29(2), 75–91 (1981)
[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 (2019)
[12] Harutyunyan, N., Bauer, A., Riehle, D.: Understanding Industry Requirements for FLOSS Governance Tools. In: IFIP International Conference on Open Source Systems, 151-167 (2018)
[13] Harutyunyan, N., Riehle, D., & Sathya, G.: Industry Best Practices for Corporate Open Sourcing. In Proceedings of the 53rd Hawaii International Conference on System Sciences (2020)
[14] Harutyunyan, N., Riehle, D.: Getting started with open source governance and compliance in companies. In Proceedings of the 15th International Symposium on Open Collaboration. ACM, 1-10 (2019)
[15] Harutyunyan, N., Riehle, D.: Industry best practices for open source governance and component reuse. In Proceedings of the 24th European Conference on Pattern Languages of Programs, 1-14 (2019)
[16] Hauge, Ø., Ayala, C., & Conradi, R.: Adoption of open source software in software-intensive organizations-a systematic literature review. Information and Software Technology, 52(11), 1133–1154 (2010)
[17] 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, 11(2) (2010)
[18] Koltun, P.: Free and open source software compliance: An operational perspective. IFOSS L. Rev., 3 (2011)
[19] Link, C.: Patterns for the commercial use of open source: legal and licensing aspects. In Proceedings of the 15th European Conference on Pattern Languages of Programs. ACM (2010)
[20] López, L., Costal, D., Ayala, C. P., Franch, X., Annosi, M. C., Glott, R., & Haaland, K.: Adoption of oss components: a goal-oriented approach. Data & Knowledge Engineering, 99, 17–38 (2015)
[21] Radcliffe, M., Odence, P.: The 2017 Open Source Year in Review. In: Black Duck Software, DLA Piper. (self-published presentation) (2017)
[22] Riehle, D., Harutyunyan, N.: Open-Source License Compliance in Software Supply Chains. In Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability. Springer, 83-95 (2019)
[23] Riehle, D.: Lessons Learned from Using Design Patterns in Industry Projects. In: Transactions on Pattern Languages of Programming II, LNCS 6510. Springer-Verlag, 1-15 (2011)
[24] Ruffin, C., Ebert, C.: Using open source software in product development: A primer. In: IEEE Software, 21(1), 82-86 (2004)
[25] Weiss, M.: Profiting even more from open source. In Proceedings of the 16th European Conference on Pattern Languages of Programs. ACM (2012)
[26] Weiss, M.: Profiting from open source. In Proceedings of the 15th European Conference on Pattern Languages of Programs. ACM (2010)