A Best Practices Handbook for Open Source Governance and Compliance

Prof. Dr. Dirk Riehle, Bayave GmbH

Date: 2017-08-05, time: 17:41:09
Status: In work.

Executive Summary

This is a handbook of best practices for establishing and managing open source governance and compliance at a company. It is in draft status and probably will keep evolving for some time.

Table of Contents

1 INTRODUCTION....6
1.1 Structure of the Handbook....6
2 HANDBOOK OVERVIEW....7
2.1 General Terminology....7
2.2 Open Source Terminology....7
2.3 Open Source Project Roles....8
2.4 Firm-internal Product Roles....8
3 GETTING STARTED....9
3.1 Start as soon as possible....9
3.2 Forbid all ungoverned use....9
3.3 Product Analysis....9
3.3.1 Create product architecture model....9
3.3.2 Use Analysis....9
3.3.2.1 Assess use by survey....9
3.3.2.2 Assess use by code scanning....9
3.3.3 Contribution Analysis....9
3.3.3.1 Assess contributions by survey....10
3.3.3.2 Assess contributions by external tools....10
3.3.4 Use one mandatory survey for initial assessment....10
3.4 IP-at-Risk Analysis....10
3.4.1 License Compliance Analysis....10
3.4.2 Risk Exposure Anlaysis....10
3.4.3 IP Risk Mitigation....10
3.4.3.1 Replace problematic components....10
3.4.3.2 Decouple problematic components....10

3.4.4 Security Risk Analysis....10
3.4.5 Capabilities Analysis....10

4 GOVERNANCE....11

4.1 Governance Management....11
4.1.1 Define goals of governance....11
4.1.2 Establish an open source program....11
4.1.3 Establish an open source program office....11
4.1.4 Define role of legal counsel....11
4.1.5 Give legal counsel veto right....11
4.1.6 Give arbitration committee decision right....11
4.1.7 Integrate program office in product development....11
4.1.8 Integrate program office in mergers and acquisitions....11
4.2 Open Source Program Office....12
4.2.1 Define roles, responsibilities, and policies....12
4.2.2 Provide roles, responsibilities, and policies in written form....12
4.2.3 Match policies to actual risks....12
4.2.4 Provide contact for internal inquiries....12
4.2.5 Provide channel for whistleblowing....12
4.2.6 Provide contact for external inquiriese....12
4.2.7 Collaborate with legal counsel on license interpretation....12
4.2.8 Track industry best practices and standards....12
4.2.9 Network to learn from others....12
4.2.10 Engage with community....12
4.3 License Interpretation....13
4.3.1.1 Use standard license interpretation....13
4.3.1.2 Develop standard license interpretation....13
4.3.1.3 Use standard license compatibility matrix....13
4.3.1.4 Develop standard license compatibility matrix....13
4.4 Component Search....13
4.4.1 Define component requiremnts....13
4.4.2 Check component repository....13
4.4.3 Follow search recommendations....13

4.5 Component Approval....13
4.5.1 Define component approval processes....14
4.5.2 Provide approval request templates....14
4.5.3 Analyse code for license compliance....14
4.5.4 Review use in context of product architecture....14
4.5.5 Add decision to component repository....14
4.6 Component Repository....14
4.6.1 Maintain a component repository....14
4.6.2 Provide component repository an single well-defined location....14
4.6.3 List both accepted and rejected components....14
4.6.4 Provide all relevant meta-data for component....14
4.6.5 Track prior approval data for reuse....14
4.7 Supplier Management....15
4.7.1 Manage suppliers....15
4.7.2 Supplier Contracts....15
4.7.2.1 Require governance certification....15
4.7.2.2 Require governance maturity....15
4.7.2.3 Require fulfillment of component requirements....15
4.7.2.4 Require bill-of-materials....15
4.7.2.5 Require use of bill-of-materials standards....15
4.7.2.6 Enable surprise audits....16
4.7.3 Integrate use-approval into delivery review....16
4.7.4 Audit suppliers....16
4.8 Component Integration....16
4.8.1 Update product architecture model....16
4.9 Contribution Management....16
4.10 Component Monitoring....16
4.10.1 Monitor component for vulnerabilities....16
4.10.2 Monitor project for community health....16
4.11 Engineering Management....17
4.11.1 Scan code contribution before commit....17
4.11.2 Maintain product architecture model....17
4.11.3 Avoid being engineering-smart but legally stupid....17

4.12 Communication.....17
4.12.1 Communicate goals of open source use.....17
4.12.2 Communicate risks of open source use.....17
4.12.3 Communicate governance processes.....17
4.12.4 Communicate component requirements.....18
4.12.5 Communicate trustworthy component sources.....18
4.13 Education.....18
4.13.1 Educate new developers.....18
4.13.2 Repeat education in regular intervals.....18
5 COMPLIANCE.....19
5.1 License Compliance.....19
5.1.1 Distribution Preparation.....19
5.1.1.1 Determine compliance artifacts.....19
5.1.1.2 Generate compliance artifacts.....19
5.1.2 Distribution Enactment.....19
5.1.2.1 Provide compliance artifacts.....19
6 CAPABILITIES.....21
6.1 Capabilities Analysis.....21
6.2 Capabilities Building.....21
A. EXAMPLE DOCUMENTS.....22
A.1 Request for Component Approval.....22
B. REFERENCES.....23
C. LICENSE.....24

1 Introduction

This handbook provides best practices for establishing and managing open source software governance and compliance at a company. It starts with the assumption that the company has decided that denying itself the benefits of open source software is counterproductive and that it would like to engage. What to do next?

This handbook provides an answer in the form of interlinked best practice descriptions. A best practice is presented as a pattern, that is, in the form of a context, problem, solution triple as known from the patterns community. All best practices thereby show how, why, where and by who they are applicable and what might come next.

In this way, the handbook is more than a passive list of best practices: It provides active guidance for establishing and managing open source software governance and compliance at a company.

1.1 Structure of the Handbook

The handbook is structured into sections and subsections of arbitrary depth. Each section has a descriptive name that captures the domain of best practices it collects. The section hierarchy represents a hierarchical breakdown of the overall domain of interest. Sections can only be the top node or intermediate nodes in the hierarchy. All leaf nodes are best practice descriptions.

A best practice description is part of a domain, captured under a heading. A best practice description follows a pattern format and consists at least of the three sections context, problem, and solution. There may also be references, examples, and other sections. A best practice description abstracts from a single example, it always aims to represent a general truth applicable widely within its context.

Best practices link to each other. Ideally, a user of the handbook starts with the first best practice and from there works his or her way through the handbook, applying one practice after another. This sequence of applications represents the establishment of the desired processes at the company.

The handbook structure and the section hierarchy allow a user of the handbook to orient themselves and jump into the middle of where they last left off.

Not all best practices need to be applied in order to achieve the company’s goals. In fact, many times the best practice application is more like a decision tree where an analysis of the context and problems at hand indicate which of several possible best practices is to be used next.

Ideally, all links between best practices are forward links, implying that if a best practice A precedes another best practice B in the handbook, it should be applied before it.

2 Handbook Overview

2.1 General Terminology

We use the following terms, which may vary between companies:

- A or the **company** is a company which provides context and scope for inner source projects. A company may have more than one business unit.
- A **business unit** of a company is a large organizational unit of the company tasked with providing products and services to its customers within the company or outside.
- An **organizational unit** (short: org. unit) is a managerial entity within line reporting within a business unit. While a business unit may be its own legal entity, an org. unit is not.

A business unit may be its own legal entity, in case of which the company is a holding company. If a business unit is not a legal entity of its own, it is just a large org. unit of the company.

2.2 Open Source Terminology

To avoid confusion, we distinguish between an open source project community and an open source component.

- The term **open source project** is a shorthand for open source project community.
- An **open source project community** is a group of people interested in and involved in one or more open source components.
- An **open source component** is a software component (artifact) developed publicly using an open source development approach.
- **Open source software** is another term for either a single open source component or a set of open source components. In any case, it is an artifact, not a community.

There are no open source projects in a project sense, because this would be a misnomer. By definition, a project has a start date and an end date. An open source project (the community) and its components (the artifacts) have a life-cycle, but they do not have a set end date. They are better viewed as products in the software product sense as they are intended to provide their service for as long as possible.

2.3 Open Source Project Roles

The main roles of actors involved in open source projects are:

- A **foundation member** is a member of an open source foundation.

An open source project is managed by a **project management committee (PMC)**. This committee either exists explicitly, typically if the project is being developed under the guidance of an open source foundation, or implicility, if it is an open source project without such a home.

- The **PMC leader** is the person leading the PMC.
- A **PMC member** is a member of a PMC.
- An **(open source) developer** is an engineer working within an open source project.

The developer role can further be detailed into two more specific roles:

- A **contributor** is a developer contributing code to a component but not integrating it.
- A **committer** is a developer integrating contributed code to a component.

In the Linux context, a committer is also called a maintainer. These are synonyms.

2.4 Firm-internal Product Roles

The main roles of actors involved with product development in companies are:

- The **CEO** has final responsibility for the company and thereby for best open source governance and compliance; typically he or she delegates this task to a program officer.
- The **program officer** is responsible for establishing and evolving best open source governance and compliance at the company; they may be on their own or lead a team.
- The **legal counsel** is responsible for providing legal advice including license interpretation, but they are not (or should not) be responsible for business risk assessment.
- An **engineering manager** is responsible for the development and delivery of a software product or of a component thereof.
- A **software architect** is responsible for the overall structure and behavior of a software product or a component thereof. This includes open source components embedded in the product.
- A **developer** is an engineer working on a product or component thereof.

None of these roles should be surprising; however there are clear responsibilities that have to be allocated to them as part of best open source governance and compliance.

3 Getting Started

3.1 Start as soon as possible

3.2 Forbid all ungoverned use

3.3 Product Analysis

3.3.1 Create product architecture model

3.3.2 Use Analysis

3.3.2.1 Assess use by survey

3.3.2.2 Assess use by code scanning

3.3.3 Contribution Analysis

3.3.3.1 Assess contributions by survey

3.3.3.2 Assess contributions by external tools

3.3.4 Use one mandatory survey for initial assessment

3.4 IP-at-Risk Analysis

3.4.1 License Compliance Analysis

3.4.2 Risk Exposure Anlaysis

3.4.3 IP Risk Mitigation

3.4.3.1 Replace problematic components

3.4.3.2 Decouple problematic components

3.4.4 Security Risk Analysis

3.4.5 Capabilities Analysis

4 Governance

4.1 Governance Management

4.1.1 Define goals of governance

4.1.2 Establish an open source program

4.1.3 Establish an open source program office

4.1.4 Define role of legal counsel

4.1.5 Give legal counsel veto right

4.1.6 Give arbitration committee decision right

4.1.7 Integrate program office in product development

4.1.8 Integrate program office in mergers and acquisitions

4.2 Open Source Program Office

4.2.1 Define roles, responsibilities, and policies

4.2.2 Provide roles, responsibilities, and policies in written form

4.2.3 Match policies to actual risks

4.2.4 Provide contact for internal inquiries

4.2.5 Provide channel for whistleblowing

4.2.6 Provide contact for external inquiriese

4.2.7 Collaborate with legal counsel on license interpretation

4.2.8 Track industry best practices and standards

4.2.9 Network to learn from others

4.2.10 Engage with community

4.3 License Interpretation

4.3.1.1 Use standard license interpretation

4.3.1.2 Develop standard license interpretation

4.3.1.3 Use standard license compatibility matrix

4.3.1.4 Develop standard license compatibility matrix

4.4 Component Search

4.4.1 Define component requiremnts

4.4.2 Check component repository

4.4.3 Follow search recommendations

4.5 Component Approval

4.5.1 Define component approval processes

4.5.2 Provide approval request templates

4.5.3 Analyse code for license compliance

4.5.4 Review use in context of product architecture

4.5.5 Add decision to component repository

4.6 Component Repository

4.6.1 Maintain a component repository

4.6.2 Provide component repository an single well-defined location

4.6.3 List both accepted and rejected components

4.6.4 Provide all relevant meta-data for component

4.6.5 Track prior approval data for reuse

4.7 Supplier Management

4.7.1 Manage suppliers

Name	Manage suppliers
Actor	Engineering manager
Context	Your product includes not only open source components, but also third-party components that are supplied to you by other software vendors. In contrast to open source projects, you are paying for the component (license) and you are receiving it from a corporate entity. You previously → *defined (your) component requirements* and they must be met by any component, open source or not.
Problem	How to ensure that a third-party component delivery meets your requirements?
Solution	First, before you select a supplier, you may → *require governance certification* or at least → *require (a minimum) govenance maturity* of them. Once you have decided for a supplier, in any delivery contract, you should → *require fulfillment of your component requirements* and you should → *require a bill-of-materials* upon delivery for which you → *require they use a bill-of-materials standard.* Upon delivery, you have to → *ensure requirements are met* and for this, you have to → *integrate use-approval into the delivery process.* If the supplier isn't certified and having to reject a component delivery is too expensive, you may want to → *enable surprise audits* and consequently also → *perform surprise audits* as to best governance practices.

4.7.2 Supplier Contracts

4.7.2.1 Require governance certification

4.7.2.2 Require governance maturity

4.7.2.3 Require fulfillment of component requirements

4.7.2.4 Require bill-of-materials

4.7.2.5 Require use of bill-of-materials standards

4.7.2.6 Enable surprise audits

4.7.3 Integrate use-approval into delivery review

4.7.4 Audit suppliers

4.8 Component Integration

4.8.1 Update product architecture model

4.9 Contribution Management

4.10 Component Monitoring

4.10.1 Monitor component for vulnerabilities

4.10.2 Monitor project for community health

4.11 Engineering Management

4.11.1 Scan code contribution before commit

4.11.2 Maintain product architecture model

Name	Maintain product architecture model
Actor	Product architect
Context	You previously → *created (a) product architecture model.* Short delivery cycles, including continuous deployment, require that you can → *generate (and provide) compliance artifacts* (among other things) at any time.
Problem	Creating a product architecture model from scratch takes longer than may be acceptable with short delivery cycles. How to make sure you can provide compliance artifacts on a whim?
Solution	Whenever you make a change to the component architecture of the product, update the product architecture model. Do not forget to → *version (the) product architecture model.* This way, the model will always be uptodate.

4.11.3 Avoid being engineering-smart but legally stupid

4.12 Communication

4.12.1 Communicate goals of open source use

4.12.2 Communicate risks of open source use

4.12.3 Communicate governance processes

4.12.4 Communicate component requirements

4.12.5 Communicate trustworthy component sources

4.13 Education

4.13.1 Educate new developers

4.13.2 Repeat education in regular intervals

5 Compliance

5.1 License Compliance

5.1.1 Distribution Preparation

5.1.1.1 Determine compliance artifacts

5.1.1.2 Generate compliance artifacts

Name	Generate compliance artifacts
Actor	Product manager
Context	The product is getting ready for release. You previously → *determined (all relevant) compliance artifacts.*
Problem	You will have to include all relevant compliance artifacts in the upcoming release.
Solution	Generate all relevant compliance artifacts from sources. Automate the generation as much as possible to avoid mistakes in an otherwise laborsome but dull process. Once the product is ready for release, you'll have to → *provide (all relevant) compliance artifacts.*

5.1.2 Distribution Enactment

5.1.2.1 Provide compliance artifacts

Name	Provide compliance artifacts
Actor	Product manager
Context	The product is getting ready for release.
Problem	

Solution	

6 Capabilities

6.1 Capabilities Analysis

6.2 Capabilities Building

A. Example Documents

A.1 Request for Component Approval

B. References

[1] Riehle, D. (2015). The Five Stages of Open Source Volunteering. In Crowdsourcing. Li, Wei; Huhns, Michael N.; Tsai, Wei-Tek; Wu, Wenjun (Editors). Springer-Verlag, 2015, 25–38. Republished from The Five Stages of Open Source Volunteering. Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Report, CS-2014–01, March 2014. Erlangen, Germany, 2014.

C. License

This document is made available under the Creative Commons BY-NC-SA 4.0 International license. Commercial (proprietary) licenses are available upon request; please send email to info@bayave.com.