OPEN SOURCE COMPLIANCE IN THE ENTERPRISE

Sponsored by FOSSID for the Open Compliance Summit 2018

IBRAHIM HADDAD, PHD

WITH CONTRIBUTIONS FROM
SHANE COUGHLAN AND KATE STEWART

THE LINUX FOUNDATION

OPEN SOURCE COMPLIANCE IN THE ENTERPRISE

2nd Edition

Ibrahim Haddad, PhD

With contributions from
Shane Coughlan and Kate Stewart

Ibrahim Haddad, PhD

Open Source Compliance in the Enterprise

2nd Edition

Copyright © 2018 The Linux Foundation. All rights reserved.

This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review and certain other noncommercial uses permitted by copyright law.

Printed in the United States of America

First Edition, 2016
Second Edition, 2018

1 Letterman Drive
Building D
Suite D4700
San Francisco CA 94129

Contents

Chapter 1 **16**
INTRODUCTION TO OPEN SOURCE COMPLIANCE **16**
A CHANGING BUSINESS ENVIRONMENT 16
ENTER OPEN SOURCE COMPLIANCE 19
Benefits of Ensuring Open Source Compliance 20
FAILURE TO COMPLY 20
Intellectual Property Failures 22
License Compliance Problems 24
Process Failures 26
LESSONS LEARNED 27
Ensure Compliance Prior to Product Shipment/Service Launch 27
Non-Compliance is Expensive 28
Relationships Matter 29
Training is Important 29

Chapter 2 **30**
ESTABLISHING AN OPEN SOURCE MANAGEMENT PROGRAM **30**
OPEN SOURCE COMPLIANCE PROGRAM 30
Compliance Strategy 31
Inquiry Response Strategy 32
Policies and Processes 32
Compliance Teams 33
Tools 34
Web Presence 35
Education 36
Automation 37
Messaging 37
Industry Initiatives 37
COMPLIANCE CHALLENGES AND SOLUTIONS 37
Long-Term Goals versus Short-Term Execution 39
Communicating Compliance 40
Establishing a Compliant Software Baseline 41
Maintaining Compliance 43
Institutionalization and Sustainability 44

Chapter 3 **46**
ACHIEVING COMPLIANCE: ROLES AND RESPONSIBILITIES **46**
OPEN SOURCE REVIEW BOARD (OSRB) 50
LEGAL 53
ENGINEERING AND PRODUCT TEAMS 54
COMPLIANCE OFFICER 56
OPEN SOURCE EXECUTIVE COMMITTEE 58
DOCUMENTATION 58
LOCALIZATION 59
SUPPLY CHAIN 59
IT 60
CORPORATE DEVELOPMENT 60

Chapter 4 **62**
OPEN SOURCE COMPLIANCE PROCESS **62**
EFFECTIVE COMPLIANCE 63
ELEMENTS OF AN END-TO-END COMPLIANCE PROCESS 64
Step 1 – Identification of Open Source 65
Step 2 – Auditing Source Code 67
Step 3 – Resolving Issues 70
Step 4 – Reviews 70
Step 5 – Approvals 72
Step 6 – Registration 73
Step 7 – Notices 74
Step 8 – Pre-Distribution Verifications 75
Step 9 – Distribution 76
Step 10 – Final Verifications 76

Chapter 5 **78**
COMPLIANCE PROCESSES AND POLICIES **78**
POLICY 78
PROCESS 79
Source Code Scan 79
Identification and Resolution 81
Legal Review 81
Architecture Review 82
Final Review 82

PROCESS STAGES' INPUTS AND OUTPUTS 83
Source Code Scan Phase 84
Identification and Resolution Phase 85
Legal Review Phase 85
Architecture Review Phase 87
Final Approval Phase 87
DETAILED USAGE PROCESS 88
INCREMENTAL COMPLIANCE PROCESS 93
OSRB USAGE FORM 95
Rules Governing the OSRB Usage Form 98
AUDITING 99
SOURCE CODE DISTRIBUTION 100
Distribution Incentives 100
Distribution Policy and Process 101
Distribution Methods and Modes 103
Distribution Checklists 104
Pre-Distribution Checklist 105
Post-Publication Checklist 106
Written Offer 107

Chapter 6 109
RECOMMENDED PRACTICES 109
COMPLIANCE PROCESS 109
Identification Phase 109
Source Code Auditing 110
Resolving Issues 112
Architectural Review 112
Approvals 113
Notices 114
Verifications 114
TOOLS AND AUTOMATION 116
Source Code Identification Tools 116
Project Management Tools 117
Software Bill of Material (BOM) Difference Tools 117
Linkage Analysis Tool 118

CHAPTER 7 120
MANAGING COMPLIANCE INQUIRIES 120

RESPONDING TO COMPLIANCE INQUIRIES 121
Acknowledge 121
Inform 122
Investigate 122
Report 122
Close Inquiry 122
Rectify 123
Improve 123
General Considerations 123
Enforcement Activities With Varying Motivations 123

CHAPTER 8 125
OTHER COMPLIANCE-RELATED PRACTICES 125
EMPLOYEE APPRAISAL 125
WEB PORTALS 126
MESSAGING 126
TRAINING 126
Informal Training 127
Formal Training 127
SOURCE CODE MODIFICATION CONSIDERATIONS 127
NOTICES CONSIDERATIONS 128
DISTRIBUTION CONSIDERATIONS 129
USAGE CONSIDERATIONS 129
ATTRIBUTION CONSIDERATIONS 131
Attribution Types 131
Presentation of Attributions 132
SPECIFIC LICENSE OBLIGATIONS 133
GENERAL GUIDELINES 134

Chapter 9 136
SCALING OPEN SOURCE LEGAL SUPPORT 136
PRACTICAL LEGAL ADVICE 136
LICENSE PLAYBOOKS 137
LICENSE COMPATIBILITY MATRIX 138
LICENSE CLASSIFICATION 140
SOFTWARE INTERACTION METHODS 142
CHECKLISTS 144
CONCLUSION 145

Chapter 10 **146**
THE OPENCHAIN PROJECT **146**
THE BUSINESS CASE FOR COMPLIANCE 146
PROCESSES ACROSS ORGANIZATIONS 146
THE PLACE OF THE OPENCHAIN PROJECT 147
DEFINING KEY REQUIREMENTS OF QUALITY OPEN SOURCE COMPLIANCE PROGRAMS 148
PROVIDING AN AVENUE TO CHECK CONFORMANCE WITH KEY PROCESSES 149
SUPPORTING CONFORMANCE WITH EDUCATIONAL MATERIAL 150
ENCOURAGING ADOPTION ACROSS MULTIPLE MARKET SEGMENTS 152
GETTING INVOLVED 152

Chapter 11 **154**
SOFTWARE PACKAGE DATA EXCHANGE® (SPDX®) **154**
INTRODUCTION 154
SPDX License List 155
SPDX License IDs 158
SPDX Specification – Background 160
Overview of an SPDX Document 161
Document Creation Information 163
Package Information 164
File Information 166
Snippet Information 167
Other Licensing Information 168
Relationships 169
Annotations 171
Tools and Other Resources for Sharing SPDX Documents 171
Tools That Can Generate SPDX Documents 172
Tools Able to Import SPDX Documents 173
Help Improve SPDX 173

Chapter 12 **175**
EVALUATING SOURCE CODE SCANNING TOOLS **175**
KNOWLEDGE BASE 175
DETECTION CAPABILITIES 175
EASE OF USE 176
OPERATIONAL CAPABILITIES 177

INTEGRATION CAPABILITIES 178
SECURITY VULNERABILITY DETECTION CAPABILITIES 178
COST 179
OTHER METRICS 180
CONCLUSION 181

Chapter 13 183
OPEN SOURCE AUDITS IN MERGER AND ACQUISITION TRANSACTIONS 183
INTRODUCTION 183
COMMON OPEN SOURCE USAGE SCENARIOS 183
INCORPORATION 184
LINKING 185
MODIFICATION 186
OPEN SOURCE AUDITS 187
AUDIT METHODS 189
SECURITY AND VERSION CONTROL 195
PRE- AND POST-ACQUISITION REMIDIATION 195
PREPARING FOR AN AUDIT AS THE ACQUISITION TARGET 196
PREPARING FOR AN AUDIT AS THE ACQUIRING COMPANY 198
CONCLUSION 200
REFERENCES 202

PREFACE

My involvement with open source compliance started early in my career as a software developer at Ericsson Research, and it has been a part of my job directly or indirectly for two decades now. Throughout my journey working with open source software, it was difficult to find practical references on open source compliance. My interest grew in making my own experiences available, first as a software developer and then as an engineering manager, so that others could possibly learn from them and then publish their own experiences. The goal is that as an industry, we can all strive towards better ways to achieve open source compliance, while minimizing impact on engineering resources and product delivery timelines.

This book summarizes my experience driving open source compliance activities in the enterprise, and focuses on practical aspects of creating and maintaining open source compliance programs. Since most of my experience was focused in the embedded space (with C and C++ being the dominant programming languages), this emphasis comes across throughout this book.

I hope you find this material useful in your day-to-day drive to achieve open source compliance. To send corrections and suggestions for improvements, please fill out the form available from **ibrahimatlinux.com/contact.html**. I will ensure that future revisions of this book will include your feedback with proper attributions.

I also offer my deepest and most sincere gratitude to The Linux Foundation's leadership and staff for their trust and their support in making this book a reality. I am very thankful to Kate Stewart and Shane Coughlan for contributing the SPDX and OpenChain chapters (respectively). It was a pleasure to collaborate with them.

WHAT'S NEW IN THIS EDITION?

The first edition of this book was published in the summer of 2016. Since then, there has been a lot of progress in many open source compliance efforts and we reached the point where we believed a refresh was needed. This second edition includes four new chapters:

- Software Package Data Exchange ® (also referred to as SPDX®): SPDX is an open standard developed under the auspices of the Linux Foundation whose goal is to provide a common method to communicate software bill of material information including components, licenses, copyrights, and security reference. The author of this chapter is Kate Stewart, Director of Strategic Projects at the Linux Foundation and one of the co-founders of the SPDX project.

- OpenChain: The OpenChain project identifies key recommended processes for effective open source management. It consists of three parts: A set of specification, a self-certification noting conformance with the specifications and a training curriculum. The author of this chapter is Shane Coughlan, Director of OpenChain at the Linux Foundation.

- Evaluating source code scanning tools: This new chapter examines metrics by which we can evaluate and compare source code scanning and license identification tools.

- Open source audits in merger and acquisition transactions: This chapter is based on an ebook (**https://www.linuxfoundation.org/resources/open-source-audits-merger-acquisition-transactions/**) we published in early 2018. It provides an overview of the open source audit process in an M&A transactions and describes in details three audit methods often used in such a scenario.

In addition to the new chapters, all other chapters from the first edition were updated to reflect current practices and in some cases incoming feedback. It is important to note that neither the author nor the contributors are legal counsels and nothing in this book should be considered as offering legal advice.

Foreword

Open source has expanded not only from an idealistic movement led by individuals around software and intellectual property but from one where organizations (e.g., governments, companies, and universities) realize that open source is a key part of their IT strategy and want to participate in its development. Early success in Linux and other open source technologies has spread to all areas of technology.

More traditional organizations are also taking notice; they are making open source software a priority and using the software for strategic advantage in their operations.

"Open Source First: Simply put, any solution developed using taxpayer dollars should be in the taxpayer's domain (open source). At GSA, we believe that all code we developed should be shared under an open license so others may benefit from it. In addition, we will give priority to using open source software as we design now solutions."
Office of the CIO, U.S. General Services Administration
(U.S. agency that oversees $66 billion of procurement annually)

"The development of Blockchain technology has the potential to redefine the operations and economics of the financial services industry. It emerges at an important time, as the industry strives to be leaner, more efficient, and more digital. Open source development will accelerate the innovation and help drive the scalability of this technology, and we are proud to support the Hyperledger Project."
Richard Lumb, Chief Executive, Financial Services, Accenture

"From increasing member investments to a growing, vibrant developer community, the Dronecode Project's first year has been extremely exciting. By bringing efforts together to establish a common platform and utilizing open source best practices, we're able to build the foundation for a new era of drone applications that extend from the camera to the cloud. The Dronecode 'full-stack' platform approach, combined with the hardware and software innovations of its members, will bring about a new generation of drones that are autonomous, aware of their environments, and continuously connected — an airborne Internet of Things."

Chris Anderson, CEO, 3DR

(Former Editor in Chief of Wired magazine and author of "The Long Tail")

"Open source is essential to our development process. It's a powerful approach that lets people work together to build great solutions while realizing shared benefits."

Rob Alexander, CIO, Capital One

Organizations are looking for guidance on how best to participate appropriately in open source communities and to do so in a legal and responsible way. Participants want to share their code and IP, and they need a trusted neutral home for IP assets (trademark, copyright, patents). They also need a framework to pool resources (financial, technical, etc.).

Participants need expertise to train them how to collaborate with their competitors in an effective manner. To that end, this book is geared to creating a shared understanding on the best ways to create shared value and innovation while adhering to the spirit and legal particulars of open source licensing.

Chapter 1

INTRODUCTION TO OPEN SOURCE COMPLIANCE

A CHANGING BUSINESS ENVIRONMENT

Traditionally, platforms and software stacks were implemented using proprietary software, and consisted of various software building blocks that originated because of internal development or via third party commercial software providers with negotiated licensing terms. The business environment was predictable and companies mitigated potential risks through license and contract negotiations with the software vendors. It was easy to identify the provider of every software component in the stack. Figure 1 illustrates the major building blocks of a traditional hardware and software platform.

Figure 1. A simplified architecture of a platform that relies on proprietary software building blocks

Over time, companies started to incorporate open source software into their platforms and software stacks due to the advantages it offers. The reasons varied from product to product, but the common theme across industries was that open source components provided compelling features out of the box, there were meaningful economies to be gained through distributed development that resulted in a faster time-to-market, and they offered a newfound ability to customize the source code. As a result, a new multi-source development model began to emerge.

Under the new model, a product could now have any combination of:

- Proprietary code, developed by the company building the product/service, with the high likelihood of these proprietary component containing open source code
- Proprietary code, originally developed by the company, then integrated in open source components but not contributed back to the upstream open source projects
- Third party commercial code, developed by third party software providers. Such code or software components are received by the company building the product/service under a commercial license, with high likelihood of it containing open source code
- Open source code, developed by the open source community and received by the company building the product/service under an open source license.

Figure 2 (next page) illustrates the multi-source development model and the various combinations of sources for incoming source code. Under this development model, software components consist of source code originating from any number of different sources and be licensed under different licenses; for instance, software component A can include proprietary source code in addition to third party proprietary source code, while software component B can include proprietary source code in addition to source code from an open source project. As the amount of open source

software grew in what were once straightforward proprietary software stacks, the business environment diverged from familiar territory and corporate comfort zones.

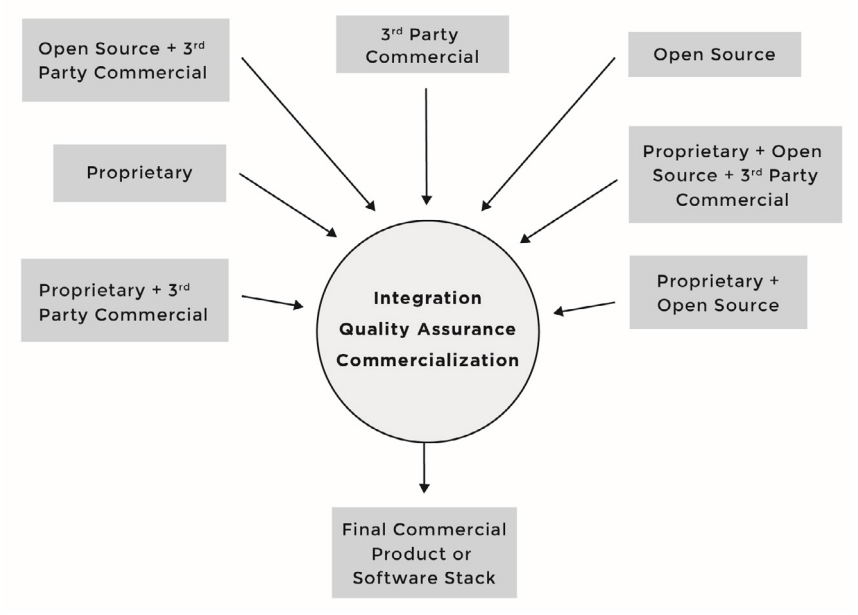


Figure 2. Multi-source development model

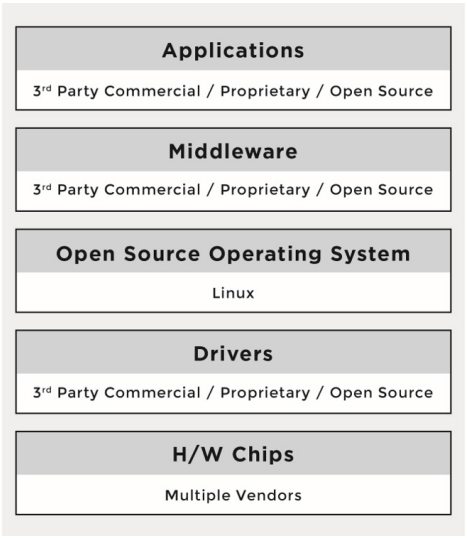


Figure 3. A simplified architectural view of a modern software platform – open source has proliferated every building block

Figure 3 illustrates the adoption of open source software throughout the various levels of a given platform or software stack.

One of the major differences between the proprietary and the multi-source development models has been that the licenses of open source software are not negotiated. There are no contracts to sign with the software providers (i.e., open source developers or projects). Rather, the individuals who initiate the project chose an open source license, and once a project reaches a certain scale, the licenses are virtually impossible to change.

When using the multi-source development model, companies must understand the implications of tens of different licenses (and combinations of licenses) coming from hundreds or even thousands of licensors or contributors (copyright holders). As a result, the risks companies previously managed through company-to-company license and agreement negotiations are now managed through robust compliance programs and careful engineering practices.

ENTER OPEN SOURCE COMPLIANCE

Open source initiatives and projects provide companies and other organizations with a vehicle to accelerate innovation through collaboration with the hundreds and sometimes thousands of communities that represent the developers of the open source software. However, there are important responsibilities accompanying the benefits of teaming with the open source community: Companies must ensure compliance with the obligations that accompany open source licenses.

Ensuring open source compliance is the process by which users, integrators, and developers of open source software observe copyright notices and satisfy license obligations resulting from the use of open source software. A well-designed open source compliance program should simultaneously ensure compliance with the terms of open source licenses and help companies protect their own intellectual property and that of third party suppliers from unintended disclosure and/or other consequences.

Open source compliance helps achieve four main objectives:

- Comply with open source licensing obligations.
- Facilitate effective use of open source in commercial products.
- Comply with third party software supplier contractual obligations.
- Protect your intellectual property from unintended disclosure.

Benefits of Ensuring Open Source Compliance

There are several benefits to achieving open source compliance. Companies that maintain a compliance program often gain a technical advantage, since compliant software portfolios are easier to service, test, upgrade, and maintain. Compliance activities can also help identify crucial pieces of open source that are in use across multiple products and parts of an organization, and/or are strategic and beneficial to that organization. Conversely, compliance can demonstrate the costs and risks associated with using open source components, as they will go through multiple rounds of review.

A healthy compliance program can deliver major benefits when working with external communities as well. In the event of a compliance challenge, such a program can demonstrate an ongoing pattern of acting in good faith.

Finally, there are less common ways in which companies benefit from strong open source compliance practices. For example, a well-founded compliance program can help a company be ready for a possible acquisition, sale, or new product or service release, where open source compliance assurance is a mandatory practice before the completion of such transactions. Furthermore, there is the added advantage of verifiable compliance in dealing with OEMs and downstream vendors.

FAILURE TO COMPLY

Throughout the software development, errors and limitations in processes can lead to open source compliance failures. Examples of such failures include:

- **Failure to provide a proper attribution notice.** An attribution notice is usually provided as a text file together with the open source component that provides acknowledgement as supplied by the contributors of open source components.

- **Neglecting to provide a license notice.** A license notice is a file that includes the open source license text included in the product or stack and is typically provided with product documentation and/or within the product or application user interface.

- **Omission of a copyright notice.** A copyright notice is an identifier placed on copies of the work to inform the world of copyright ownership.

- **Failure to provide a modification notice.** A modification notice calls out modifications to the source code in a change log file, such as those required by the GPL and LGPL. An example of a modification notice is shown below:

```
/*
* Date         Author          Comment
* 10/15/2015   Ibrahim Haddad  Fixed memory leak in nextlst()
*/
```

- **Making inappropriate or misleading statements** in the product documentation or product advertisement material.

- **Failure to provide the source code.** Making source code available (including the modifications) is one of the requirements of the GPL/LGPL family of licenses.

- **Failure to provide a written offer for example when using GPL/LGPL license source code.** A written notice provides the end users of the product with information on open source software included in the product, in addition to information on how to download source code that is eligible to distribution in fulfillment of license obligations. It is usually provided as part of the product documentation and accessible from the product's user interface. A basic example of a written offer would look like:

To obtain a copy of the source code being made publicly available by FooBar, Inc. related to software used in this product, you can visit http://opensource.foobar.com or send your request in writing by email to opensource@foobar.com or by regular postal mail to:

FooBar Inc.
Open Source Program Office Street Address
City, State, Postal Code Country

- **Failure to provide the build scripts** needed to compile the source code (per GPL and LGPL family of licenses).

In the following sections, we will explore three types of compliance failures, discuss how they occur, how to avoid them and how to prevent them from recurring.

Intellectual Property Failures

Table 1 (next page) provides examples of common accidental admixture of proprietary and open source IP that can arise during the software development process leading to license compliance issues. These problems most commonly involve mixing source code that is licensed under incompatible or conflicting licenses (e.g., proprietary, third party, and/or open source).

Such admixtures may force companies to release proprietary source code under an open source license, thus losing control of their (presumably) high-value intellectual property and diminishing their capability to differentiate in the marketplace.

The intellectual property failures can lead to one or more of the following results:

- An injunction preventing a company from shipping the product until the compliance issue has been resolved
- A requirement to distribute proprietary source code that corresponds to the binaries in question under an open source license (depending on the specific case)

- A significant re-engineering effort to eliminate the compliance issues
- Embarrassment with customers, distributors, third party proprietary software suppliers and an open source community

Table 1. Examples of intellectual property failures with examples on how to discover them and how to avoid them

Problem Type	How Discovered	How to Avoid
Inserting open source code into proprietary or 3rd party code Occurs during development process when developers copy/paste open source code (aka "snippets") into proprietary or 3rd party source code	By scanning the source code for possible matches with open source code	Offer training to increase awareness of compliance issues, open source (OS) licenses, implications of including OS code in proprietary or 3rd party code Conduct regular code scans of all project source code for unexpected licenses or code snippets. Require approval to use OS software before committing it into product repository
Linking of open source into proprietary source code (or vice versa – specific to C/C++ source code) Occurs as a result of linking software components that have conflicting or incompatible licenses	With a dependency-tracking tool that allows discovery of linkages between different software components; ID if type of linkage is allowed per company's OS policies	Offer training on linkage scenarios based on company compliance policy Regularly run dependency tracking tool to verify all linkage relationships; flag any issues not in line with compliance policies
Inclusion of proprietary code into an open source component Occurs when developers copy/paste proprietary source code into OS software	By scanning source code. Tool will ID source code that doesn't match what's provided by OS component, triggering various flags for Audit	Train the staff Conduct regular source code inspections Require approval to include proprietary source code in OS components

License Compliance Problems

License compliance problems are typically less damaging than intellectual property problems, as they do not have the side effect of forcing you to release your proprietary source code under an open source license. License compliance failures may result in any (or a combination) of the following:

- An injunction preventing a company from shipping a product until they release the source code.
- Support or customer service headaches due version mismatches between the binary in the product and source code released. There are many reported incidents individuals calling support hotlines to inquire about open source code releases. In some cases, such incidents pushed companies to provide basic training on open source compliance to their hotline or tech support staff.
- Embarrassment and/or bad publicity with customers and open source community.

Table 2. Examples of license compliance problems with suggestions on how to avoid them

Problem Type	How to Avoid
Failure to publish source code packages as part of meeting license obligations	Follow a detailed compliance checklist to ensure that all compliance action items have been completed when a given product, application, or software stack is released into the market
Failure to provide correct version of the source code corresponding to the shipped binaries	Add a verification step into the compliance process to ensure that you're publishing the version of source code that exactly corresponds to the distributed binary version
Failure to release modifications that were introduced to the open source software being incorporated into the shipping product	Use a bill of material (BOM) difference tool that allows the identification of software components that change across releases Re-introduce the newer version of the software component in the compliance process Add the "compute diffs" of any modified source code (eligible for open source distribution) to the checklist item before releasing open source used in the product
Failure to mark open source code that has been changed or to include a description of the changes	Add source code marking as checklist item before releasing source code Conduct source code inspections before releasing the source code Add milestone in compliance process to verify modified source code has been marked as such Offer training to staff to ensure they update the change logs of source code files as part of the development process

Table 2 above provides examples of the most common license compliance problems that occur during the software development process, and offers tips on how to avoid them.

Process Failures

Process failures can lead to infringement of the open source licensing terms such as the inability to meet the license obligations.

Table 3. Sample process compliance failures with suggestions on how to avoid them

Failure	How to Avoid
Failure of developers to request approval from the internal open source committee (often called Open Source Review Board) to use open source software, or Failure to submit a request within a reasonable period of time	Train employees on policies and processes Conduct periodic full scans of all software to detect any OS not corresponding to a given approval form. If OS component or snippet is found in the build system without a corresponding compliance ticket, a new ticket is auto-generated. Include compliance in performance reviews; e.g., failure to abide by the compliance policies directly affects employees' bonuses Mandate that developers file approval requests early, even if they didn't yet decide on adoption of OS code
Failure to take the open source training	Ensure OS training is part of the employees' professional development plan and that it is monitored as part of their performance review process
Failure to audit the source code	Provide proper training to compliance staff Conduct periodic source code scans Ensure that auditing is a milestone in the iterative development process Provide proper level of staffing to the auditing team
Failure to resolve the audit findings	Do not allow compliance tickets to be resolved if audit report is not finalized. Compliance ticket is closed only if no open subtasks are attached to it

Table 3 (previous page) lists the most common compliance process failures that occur during the stages of the software development process, and discusses how to avoid them.

LESSONS LEARNED

In the past few years, we have witnessed several cases of non-compliance that made their way to the public eye. Increasingly, the legal disposition towards non-compliance has lessons to teach open source professionals — lessons that we will explore in following subsections.

Ensure Compliance Prior to Product Shipment/Service Launch

The most important outcome of non-compliance cases has been that the companies involved ultimately had to comply with the terms of the license(s) in question, and the costs of addressing the problem after the fact has categorically exceeded those of basic compliance. Therefore, it is really a smart idea to ensure compliance before a product ships or a service launches.

It is important to acknowledge that compliance is not just a legal-department exercise. All facets of the company must be involved in ensuring proper compliance and contributing to correct open source consumption and, when necessary, redistribution. This involvement includes establishing and maintaining consistent compliance policies and procedures as well as ensuring that the licenses of all the software components in use (proprietary, third party, and open source) can co-exist before shipment or deployment. To that effect, companies need to implement an end-to-end open source management infrastructure that will allow them to:

- Identify all open source used in products/services, and/or used internally
- Perform architectural reviews to verify if and how license obligations are extending to proprietary and third party software components
- Collect the applicable open source licenses for review by Legal

- Develop open source use and distribution policies and procedures
- Mitigate risks through architecture design and engineering practicess

Non-Compliance is Expensive

Most of the public cases related to non-compliance have involved GPL source code. Those disputes reached a settlement agreement that included one or more of these terms:

- Take necessary action to become compliant. In extreme cases, this correction can be very costly in terms of engineering effort and impact to the product launch timeline.
- Appoint a Compliance Officer to establish a formal compliance program, monitor and ensure compliance on an ongoing basis.
- Notify previous recipients of the product with the non-compliance issue that the product contains open source software and inform them of their rights with respect to that software.
- Publish licensing notice on company website.
- Provide additional notices in product publications.
- Make available the source code including any modifications applied to it (specific to the GPL/LGPL family of licenses and similar licenses with a code distribution requirement).
- Cease binary distribution of the open source software in question until it has released complete corresponding source code or make it available to the specific clients affected by the non-compliance.
- In some cases, pay an undisclosed amount of financial consideration to the plaintiffs.

Furthermore, companies have incurred other costs when their compliance has been challenged such as:

- Discovery and diligence costs in response to the compliance inquiry, where the company had to investigate the alleged inquiry and perform due diligence on the source code in question
- Outside and in-house legal costs
- Damage to brand, reputation, and credibility

In almost all cases, the failure to comply with open source license obligations has also resulted in public embarrassment, negative press, and damaged relations with the open source community.

Relationships Matter

For companies using open source software in their commercial products, it is recommended to develop and maintain a good relationship with members of the open source communities that create and sustain the open source code they consume. The communities of open source projects expect companies to honor the licenses of the open source software they include in their products. Taking steps in this direction, combined with an open and honest relationship, is very valuable.

Training is Important

Training is an essential building block in a compliance program, to ensure that employees have a good understanding of the policies governing the use of open source software. All personnel involved with software need to understand the company's policies and procedures. Companies often provide such education through formal and informal training sessions.

Chapter 2

ESTABLISHING AN OPEN SOURCE MANAGEMENT PROGRAM

An open source management program provides a structure around all aspects of open source software, including selection, approval, use, distribution, audit, inventory, training, community engagement, and public communication.

This chapter provides a high-level overview of the various elements in an open source management program, surveys the challenges in establishing a new compliance program, and provides advice on how to overcome those challenges.

OPEN SOURCE COMPLIANCE PROGRAM

We begin this chapter with an overview of the core elements needed in a successful open source compliance program. Figure 4 (next page) provides an overview of these essential elements that we discuss throughout this chapter.

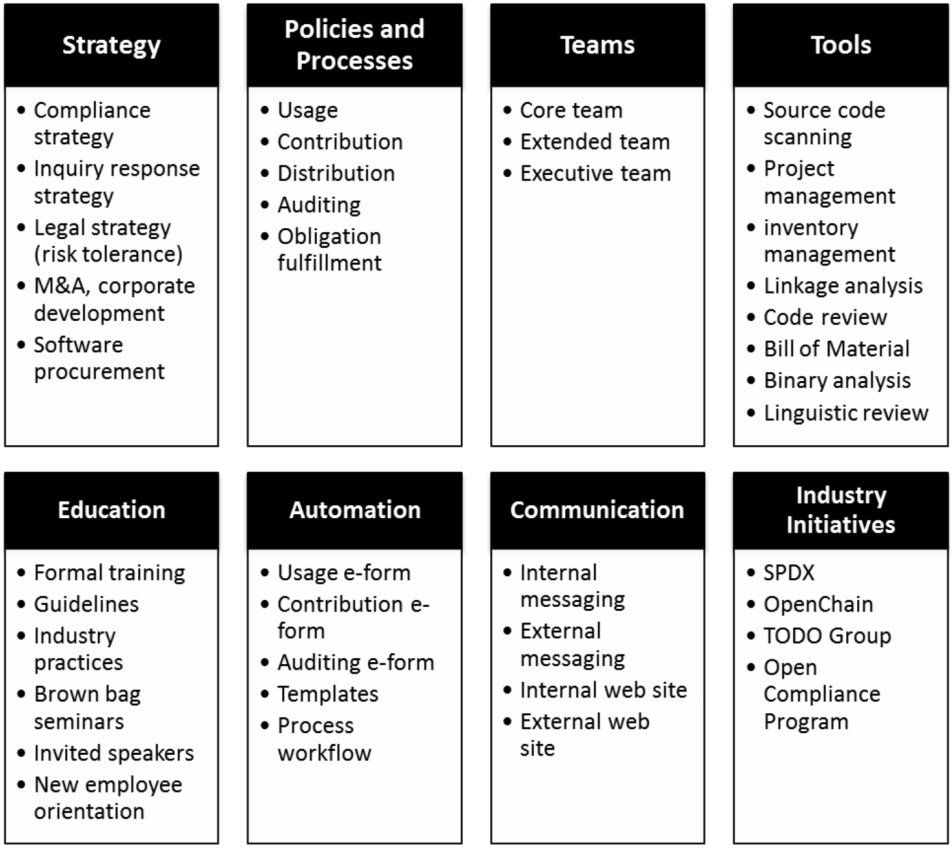


Figure 4. Essential elements of an open source management program consisting of eight essential elements: strategy, policies and processes, teams, tools, education, automation, communication and participating in relevant industry initiatives.

Compliance Strategy

The open source compliance strategy drives the business-based consensus on the main aspects of the policy and process implementation. If you do not start with that high-level consensus, driving agreement on the details of the policy and on investments in the process tends to be very hard, if not impossible.

The strategy establishes what must be done to ensure compliance and offers a governing set of principles for how personnel interact with open source software. It includes a formal process for the approval, acquisition,

and use of open source, and a method for releasing software that contains code licensed under an open source license.

Inquiry Response Strategy

The inquiry response strategy establishes what a company must do when their compliance efforts are challenged. Several companies received negative publicity — and some were formally challenged — because they ignored requests to provide additional compliance information, did not know how to handle compliance inquires, lacked or had a poor open source compliance program, or simply refused to cooperate with the inquirer. None of these approaches is fruitful or beneficial to any of the parties involved.

Therefore, companies should have a process in place to deal with incoming inquiries, acknowledge their receipt, inform the inquirer that they will be looking into it, and provide a realistic date for follow-up. Chapter 7 discusses a simple process for managing open source compliance inquiries.

Policies and Processes

The open source compliance policy is a set of rules that govern the management of open source software (both use of and contribution to). Processes are detailed specifications as to how a company will implement these rules on a daily basis. Compliance policies and processes govern the various aspects of using, contributing, auditing, and distribution of open source software.

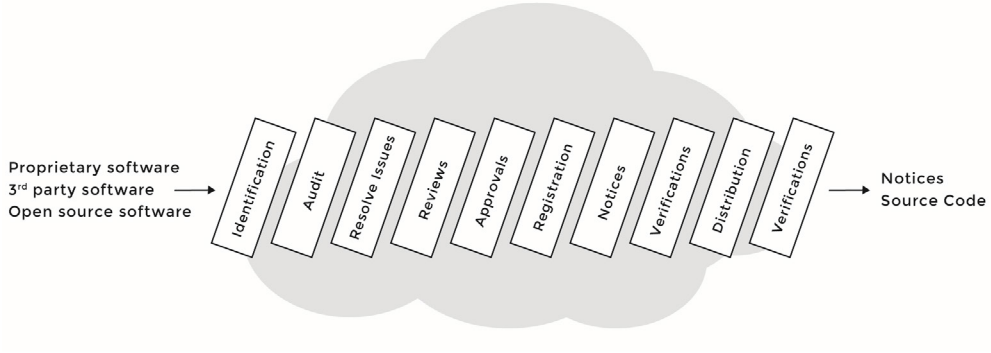


Figure 5. Sample compliance due-diligence process

Figure 5 illustrates a sample compliance process, with the various steps each software component will go through as part of the due diligence. Chapter 4 is dedicated to discussing the compliance process.

Compliance Teams

The open source compliance team is a cross-disciplinary group consisting of various individuals tasked with the mission of ensuring open source compliance. The core team, often called the Open Source Review Board (OSRB), consists of representatives from engineering and product teams, one or more legal counsel, and the Compliance Officer. The extended team consists of various individuals across multiple departments that contribute on an ongoing basis to the compliance efforts: Documentation, Supply Chain, Corporate Development, IT, Localization and the Open Source Executive Committee (OSEC). However, unlike the core team, members of the extended team are only working on compliance on a part-time basis, based on tasks they receive from the OSRB.

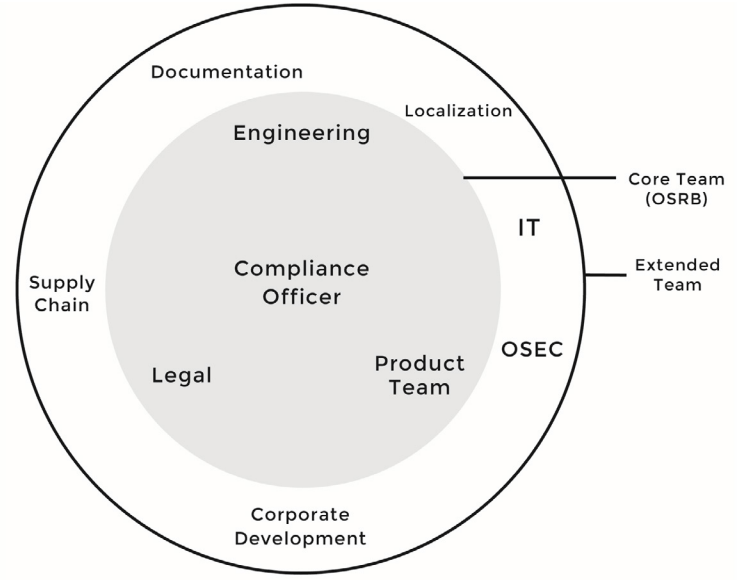


Figure 6. Individuals and teams involved in ensuring open source compliance

Figure 6 illustrates the two teams involved in achieving compliance: the core team and the extended team. Chapter 3 provides a detailed discussion on the roles and responsibilities of individuals involved in achieving open source compliance.

Tools

Open source compliance teams use several tools to automate and facilitate the auditing of source code and the discovery of open source code and its licenses. Such tools include:

- A compliance project management tool to manage the compliance project and track tasks and resources.
- A software inventory tool to keep track of every single software component, version, and product that uses it, and other related information.

- A source code scanning and license identification tool to help identify the origin and license of the source code included in the build system. Chapter 12 explores the various metrics companies can use to evaluate and compare such existing tools in the market.
- A linkage analysis tool to identify the interactions of any given C/C++ software component with other software components used in the product. This tool will allow you to discover linkages between source code packages that do not conform to company policy. The goal is to determine if any open source obligations extend to proprietary or third party software components. If a linkage issue is found, a bug ticket is assigned to Engineering with a description of the issue in addition to a proposal on how to solve the issue.
- A code review tool to review the changes introduced to the original source code before disclosure as part of meeting license obligations.
- A bill of material (BOM) difference tool to identify the changes introduced to the BOM of any given product given two different builds. This tool is very helpful in guiding incremental compliance efforts.

Web Presence

Companies use portals in two directions: inwards, inside the company; and outwards, as a window to the world and the open source community.

The internal portal hosts the compliance policies, guidelines, documents, training, announcements, and access to mailing lists. The external portal offers a public platform for the world and the open source community, as well as a venue to post source code of open source packages, notices, and other disclosures, in fulfillment of license obligations.

Education

Education is an essential building block in a compliance program, to help ensure that employees possess a good understanding of policies governing the use of open source software. The goal of providing open source and compliance training — formally or informally — is to raise awareness of open source policies and strategies and to build a common understanding around the issues and facts of open source licensing as well as the business and legal risks of incorporating open source software in products and/or software portfolios. Training also serves as a venue to publicize and promote the compliance policy and processes within the organization and to foster a culture of compliance.

Formal Training

Depending on the size of the company and the extent to which open source is included in its commercial offerings, the company can mandate that employees working with open source take formal instructor-led courses, possibly culminating in actual exams.

Informal Training

Informal training channels may include any or all of the following:

- **Brown bag seminars:** Brown bag seminars are usually presentations made during lunchtime by a company employee or an invited speaker. The goal of these brown bag seminars is to present and evoke discussions of the various aspects of incorporating open source in a commercial product or an enterprise software portfolio. These sessions can also include discussions of the company's compliance program, policies, and processes.
- **New employee orientation:** In some instances, the Compliance Officer presents on the company's compliance efforts, rules, policies, and processes to new employees as part of employee orientation, supplying new employees with necessary open source management information: who to talk to, what internal website to visit, how to sign up for open source and compliance training, etc.

Automation

Developers who wish to use or contribute to open source software will be requested to submit online requests and get proper approvals. This process is best managed via an automated online system, commonly a bug tracker that has a specifically designed workflow to accommodate the management of open source compliance.

Messaging

Messaging, both internal and external, is an integral part of any compliance program. The single most important recommendation with respect to messaging is to be clear and consistent, whether it is internally explaining the company's goals and concerns around open source to your employees or externally toward the developer communities of the open source projects you use in your product/software stack.

Industry Initiatives

There are several industry initiatives targeted to simplify and enable better open source compliance such as the Software Packaged Data Exchange® (SPDX), OpenChain, the Linux Foundation Open Compliance Program, and the TODO Group. We discuss these initiatives throughout the book. We recommend companies to get involved in these initiatives to gain additional knowledge and support their internal compliance efforts.

COMPLIANCE CHALLENGES AND SOLUTIONS

Companies will almost certainly face challenges establishing their open source compliance program. In the following sections, we explore five common challenges and offer recommendations on how to overcome them.

Creating a Compliance Program

The first challenge is to balance the compliance program and its supporting infrastructure with (existing) internal processes while meeting deadlines to

ship products and launch services. Various approaches can help ease or solve such challenges and assist in the creation of a streamlined program that does pose a burden to development activities.

Proposed Solutions

EXECUTIVE SUPPORT

Executive-level, long-term commitment to the open source management program is essential to ensure success and continuity.

LIGHTWEIGHT POLICIES AND PROCESSES

Processes and policies are important; however, they have to be light and efficient so that development teams do not regard them as overly burdensome to the development process.

Streamline open source management upon two important foundational elements: a simple and clear compliance policy and a lightweight compliance process.

MANDATE BASIC RULES

As part of establishing a compliance program, you will need to create some simple rules that everyone must follow:

- Require developers to fill out a request form for any open source software they plan to incorporate into a product of software stack.
- Require third party software suppliers to disclose information about open source software included in their deliverables. Your software suppliers may not have great open source compliance practices. Therefore, a general recommendation is for companies to update their contractual agreement including language related to open source disclosures and clarifying all matters related to open source compliance.
- Mandate architecture reviews and code inspections for the Open Source Review Board (OSRB) to understand how software components are interrelated and to discover license obligations that can

propagate from open source to proprietary software. You will need proper tooling to accommodate a large-scale operation.

- Scan all incoming software received from third party software providers and ensure that their open source disclosures are correct and complete.

INTEGRATE COMPLIANCE IN THE DEVELOPMENT PROCESS

The most successful way to establish compliance is to incorporate the compliance process and policies, checkpoints and activities as part of existing software development processes.

Long-Term Goals versus Short-Term Execution

Figure 4 described the essential elements needed for a successful compliance program. Some team members may be overwhelmed by the amount of work needed to implement such a complete program. In reality, it is not very difficult, because you do not have to implement all elements simultaneously.

The priority for all organizations is to ship products and services on time while building and expanding their internal open source compliance infrastructure. Therefore, you should expect to build your compliance infrastructure as you go, keeping in mind scalability for future activities and products. The key is thoughtful and realistic planning.

Proposed Solutions

- Plan a complete compliance infrastructure that can meet your long- term goals, but then implement only the elements stepwise, as needed for short-term execution. For instance, if you are just starting to develop a product or deliver a service that includes open source and you do not yet have any compliance infrastructure in place, the immediate concern should be establishing a compliance team, processes and policy, tools and automation, and training your employees. Having kicked off these activities (in that order) and possessing a good grip on the build system (from a compliance perspective), you can move on to other program elements.

- Establish essential policies and processes.
- Incorporate compliance milestones as part of the development process.

Communicating Compliance

Communication is essential to ensure the success of compliance activities. Companies should consider two important types of communication: internal communication within the organization, and external towards the developer communities of the open source projects used in their products and towards the end users of their products.

Internal Communication

Companies need internal compliance communication to ensure employees are aware of what is involved when they include open source in a commercial software portfolio, and to ensure that they are educated about the company's compliance policies, processes, and guidelines. Internal communications can take any of several forms:

- Email communication providing executive support
- Formal training to all employees working with open source software
- Brown-bag open source and compliance seminars to bring additional compliance awareness and promote active discussion
- An internal open source portal to host the company's compliance policies and procedures, open source related publications and presentations, mailing lists, and a discussion forum related to open source and compliance
- A company-wide open source newsletter, usually sent every other month or on quarterly basis, to raise awareness of open source compliance

External Communication

Companies need external compliance communications to ensure that the open source community is aware of their efforts to meet the license obligations of the open source software they are using in their products.

- External communications can take one of several forms:
- Website dedicated to distributing open source software for the purpose of compliance
- Outreach and support of open source organizations: Such activities are important to help the company build relationships with open source organizations, understand the roles of these organizations, and contribute to their efforts where it makes sense
- Participation in open source events and conferences: Participation can be at various levels ranging from sponsoring an event, to contributing presentations and publications, or simply sending developers to attend and meet open source developers and foster new relationships with open source community members

Establishing a Compliant Software Baseline

One of the initial challenges when starting a compliance program is to find exactly which open source software is in use and under which licenses it is available. We often refer to this initial auditing process as establishing a clean compliance baseline for the product or software stack.

The activity of establishing a compliant baseline is an intensive activity over a period of time that can extend for months, depending on how soon you started the compliance activities in parallel to the development activities.

Proposed Solutions

Organizations achieve initial compliance through the following activities:

- Early submission and review of open source usage requests.
- Continuous automated source code based on a predefined interval of time for all source code.
- Continual scans on the source code base, including code received from third party software providers, to intercept source code that is checked into the code base without a corresponding compliance

ticket. Such source code scans can be scheduled to run on a monthly basis, for instance.

- Enforced design and architectural review, in addition to code inspections, to analyze the interactions between open source, proprietary code, and third party software components. Such reviews are mandatory only when a given interaction may invoke license compliance obligations.

If a company fails to establish baseline compliance, they are guaranteed to face compliance challenges in any future revisions of the same product (or other products/services using code from the initial baseline). To guard against such scenarios, companies should consider the following:

- Implement lightweight policies and processes that set a framework for managing the intake of open source software.
- Include compliance checkpoints as part of the software development process. Ideally, with every development milestone, you can incorporate a corresponding compliance milestone, ensuring that all software components used in the build have parallel and approved compliance tickets.
- Ensure availability of a dedicated compliance team. Chapter 3 covers this topic in detail.
- Utilize tools and automation to support efficient processing of compliance tickets. We discuss this topic throughout the book. Chapter 12 focuses on various criteria to evaluate source code scanning tools.

Maintaining Compliance

There are several challenges in maintaining open source compliance, similar to those faced when establishing baseline compliance, but on a smaller, incremental scale.

Maintaining compliance is a continuous effort that depends on discipline and commitment to incorporate compliance activities into existing engineering and business processes. Figure 7 illustrates the concept of incremental compliance. Incremental compliance is the act of ensuring compliance between versions, covering the delta between the last compliant code baseline and current source code base.

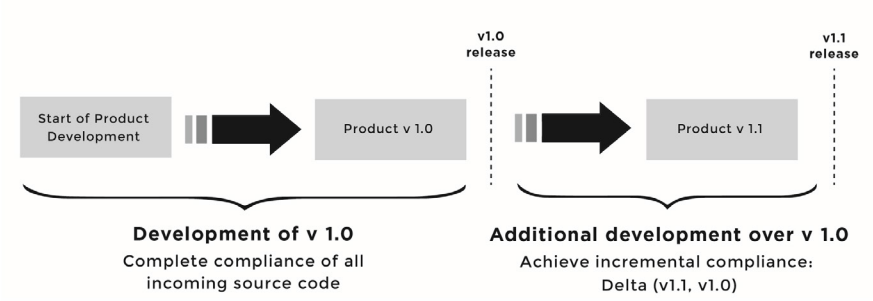


Figure 7. Example of incremental compliance

Proposed Solutions

Companies can maintain incremental compliance through the following activities:

- Adherence to the company's compliance policy and process, in addition to any provided guidelines
- Continuous audits of all source code integrated in the code base, regardless of its origins
- Continuous improvements to the tools used in ensuring compliance and automating as much of the process as possible to ensure high efficiency in executing the compliance program

Institutionalization and Sustainability

Maintaining open source compliance activities is an ongoing challenge as the organization grows and deploys more open source software in new products and services. Companies can take several steps to institutionalize compliance within their development culture and to ensure its sustainability.

Proposed Solutions

SPONSORSHIP

Executive-level commitment is essential to ensure sustainability of compliance activities. A company executive required to be the champion and financial sponsor for the open source compliance program.

CONSISTENCY

Achieving consistency across the company is key in large companies with multiple business units and subsidiaries. A consistent approach helps with recordkeeping, and facilitates sharing best practices, tooling, guidelines and source code across groups. Consistency is managing compliance is mandatory in large companies with multiple business units.

MEASUREMENT AND ANALYSIS

Measure and analyze the impact and effectiveness of compliance activities, processes, and procedures with the goal improving the compliance program. It is hard to improve a specific function if you cannot measure it.

Metrics will help you communicate the productivity advantages that accrue from each program element when promoting the compliance program.

REFINING COMPLIANCE PROCESSES

The scope and nature of an organization's use of open source is dynamic—dependent on products, technologies, mergers, acquisitions, offshore development activities, and many other factors. Therefore, it is necessary to continuously review compliance policies and processes and introduce improvements.

Furthermore, open source license interpretations, legal compliance risks, and copyright trolling continue to evolve. A compliance program must evolve as well to mitigate such risks via better and improved engineering and compliance practices.

ENFORCEMENT

An effective compliance program should include mechanisms for ongoing monitoring of adherence to the program and for enforcing policies, procedures, and guidelines throughout the organization. One way to enforce the compliance program is to integrate it within the software development process. Some companies go a little further by ensuring that a specific portion of their employees performance evaluation depend on their commitment to and execution of compliance program activities at their individual and group levels.

STAFFING

Companies must ensure they allocate proper staffing to the compliance function, and offer adequate compliance training to all employees. In larger organizations, the compliance officer and related roles are FTEs (full time equivalents); in smaller organizations, the responsibility of compliance is likely to be a shared and/or a part-time activity.

Chapter 3

ACHIEVING COMPLIANCE: ROLES AND RESPONSIBILITIES

Companies have used various ways to structure their teams responsible for fulfilling this function. Some companies have opted for a centralized team (Open Source Program Office) led by an individual who has engineering, legal and operational experience. Other companies have opted for a cross-functional team that consists of a dedicated Open Source Compliance Officer who has access to various individuals and teams that contribute to the compliance effort without being part of a centralized team. In this chapter, we examine roles and responsibilities of the individuals and teams trusted with ensuring compliance regardless of the specific structural model of these teams. A single individual, no matter how adept, cannot successfully implement open source compliance across a whole organization.

Figure 8 (next page) illustrates a breakdown of the different departments responsible for achieving open source compliance. There are two teams involved in achieving compliance: a core team and an extended team, with the latter typically being a superset of the former.

The core team, often called the Open Source Review Board (OSRB), consists of representatives from legal, engineering, and product teams, in addition to the Compliance Officer. Table 4 (next page) describes the roles and responsibilities of each participant in this core team.

The extended team, described in Table 5 (page 49), consists of various individuals across multiple departments that contribute on an on-going basis to the compliance efforts: Documentation, Supply Chain, Corporate Development, IT, Localization, and the Open Source Executive Committee (OSEC). However, unlike the core team (in substantial organizations), members of the extended team are contributing to the compliance efforts on a part- time basis, based on tasks they receive from the OSRB.

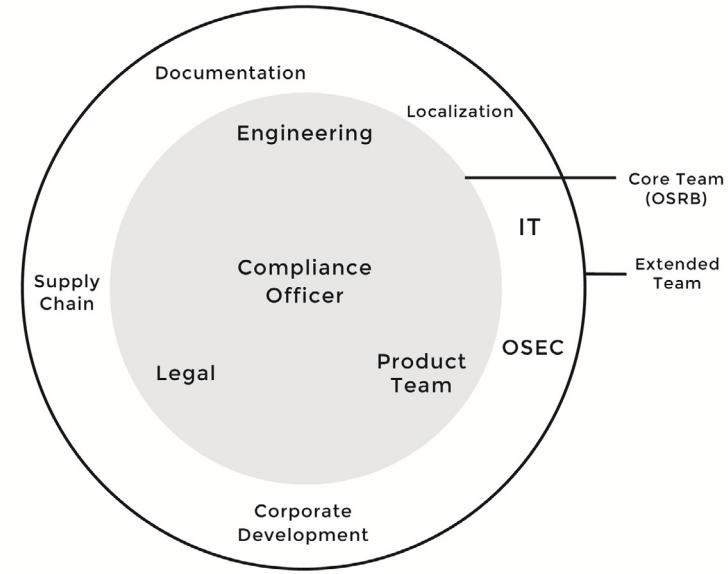


Figure 8. Individuals and teams involved in ensuring open source compliance

Table 4. Primary roles and responsibilities of the compliance core team (OSRB)

Participant	Primary Roles and Responsibilities
Legal Representative This representation varies from a Legal counsel to a Legal paralegal depending on the specific task.	Participate in OSRB and OSEC Review and approve usage, modification, distribution of open source (OS) software Provide guidance on licensing Contribute to and approve training Contribute to improving the OS compliance program Review and approve content of OS portals Review and approve list of obligations to fulfill Review and approve open source notices

Table 4 (cont'd). Primary roles and responsibilities of the compliance core team (OSRB)

Participant	Primary Roles and Responsibilities
Engineering and Product Team Representative In some companies, there is no distinction between engineering and product teams.	Participate in OSRB and OSEC Follow compliance policies and processes Integrate compliance practices in development process Contribute to improving the compliance program Follow technical compliance guidelines Respond quickly to all questions Conduct design, architecture, and code reviews Prepare software packages for distribution
Compliance Officer Open source compliance officer is not necessary a dedicated resource. In most cases, the individual fulfills the role of Manager or Director of Open Source.	Drive all compliance activities Coordinate source code scans and audits Coordinate distribution of source code packages Participate in OSRB and OSEC Contribute to compliance and OS training Contribute to improving compliance program Report to OSEC on compliance activities Contribute to creation of new tools to facilitate automation, discovery of OS in development environment

Table 5. Primary roles and responsibilities of the compliance extended team

Participant	Primary Roles and Responsibilities
Open Source Executive Committee (OSEC) Decide on open source strategy	Review and approve proposals to release IP Review and approve proposals to release proprietary source code under an open source license. This is not required if the source code was created with the assumption that it will be open sourced.
Documentation	Include open source license information and notices in the product documentation
Localization	Translate basic information in target languages about open source information related to the product or software stack
Supply Chain	Mandate third party software providers to disclose open source in licensed or purchased software components Assist with ingress of third party software bundled with and/or includes open source software
Information Technology (IT)	Provide support and maintenance for the tools and automation infrastructure used by the compliance program Create and/or acquire new tools based on OSRB requests
Corporate Development	Request open source compliance be completed before a merger or an acquisition Request open source compliance be completed when receiving source code from outsourced development centers or third party software vendors

OPEN SOURCE REVIEW BOARD (OSRB)

The OSRB is responsible for:

- Ensuring mutual compliance with third party software and open source software licenses.
- Facilitating effective usage of and contributions to open source software.
- Protecting proprietary intellectual property (and consequently product differentiation) from unintended disclosure.

On a daily basis, OSRB members are involved in the following tasks:

- Establish the Compliance End-to-End Process: The OSRB is responsible for creating the compliance end-to-end process including usage, audit, development, engagement, assurance, and compliance management. Chapter 4 provides an overview of the end-to-end compliance process.
- Create and maintain compliance policies, processes, guidelines, templates, and forms used in the compliance program.

- Review requests for the use, modification, and distribution of open source: The OSRB reviews and approves incoming requests from engineering and product teams for using open source. Chapter 6 provides a discussion on the usage process.
- Perform software audits: The OSRB performs audits on all software included in a product or a software stack, which involves the following tasks:
    - Run a source code scanning tool over the software base
    - Analyze the results provided by the scanning tool
    - Address all the hits, possible matches, and licensing conflicts flagged by the scanning tool
    - Oversee the closure of all issues identified by scanning tools
    - Create a final audit report that captures all open source software components and snippets, their origin and license.

  Note:
  Depending on the size of the organization, auditing responsibilities can be assigned to the OSRB or to an independent team (auditing team) that reports to the Compliance Officer. Chapter 6 provides a discussion of the auditing process.
- Perform architectural reviews: As part of the approval process, the OSRB performs architecture review with engineering representative to analyze the interaction between open source code, proprietary code, and third party source code. The goal of this review is to ensure that developers respect the architectural guidelines and that the interactions among open source, proprietary, and third party software are within the acceptable legal guidelines. The architecture review includes performing a linkage analysis review to determine if any open source license obligations propagate to proprietary or third party software through linking.

- Verify the resolution of issues that deter releasing product or launching services that contain open source.
- Provide guidance on open source questions coming from company staff and engineers.
- Perform code inspections as part of the pre-distribution verification, to ensure that open source license text and copyright notices have been kept intact and that engineers have updated the change logs to reflect the changes introduced to the source code.
- Compile a list of all applicable license obligations that must be fulfilled and pass it to appropriate departments for fulfillment: As part of the pre-distribution process, the OSRB performs final checks before product or service releases.
- Develop and offer open source and compliance training: The OSRB drives the development of open source and compliance training to ensure that employees have a good understanding of the company's open source policies and compliance practices. In some cases, the training also covers the most common open source licenses and the concerns arising from using open source in commercial contexts. Training must be mandatory for all staff engaged in management and development of software using open source.
- Host and maintain the company's open source websites: The internal website, intended for employees, focuses on open source processes and policies, guidelines, training, and announcements. The external website usually exists for the primary reason of making available source code packages in fulfillment of certain compliance obligations.
- Handle compliance inquiries: The OSRB is responsible for answering any inquiries sent to the company in relation to open source compliance. Chapter 9 discusses the process of handling compliance inquiries.
- Maintain records of compliance: The OSRB is responsible for ensuring the correctness and upkeep of compliance records for any given open source software component.

- Review end-user documentation to ensure that appropriate copyright, attribution, and license notices are presented to the end user of the product/service regarding open source included in the product or the software stack. In addition, specific to the GPL/LGPL family of licenses, provide a written offer on how to obtain the source code, when applicable.
- Recommend new tools to be used as part of the compliance infrastructure that will contribute to making the compliance work more efficient.
- Sign off on product distribution from an open source compliance perspective.
- Develop community involvement policy, process, procedures, and guidelines. This responsibility is not compliance-related; however, it is listed here for completion purposes.

LEGAL

The Legal Counsel is a core member of the OSRB, the committee that ensures compliance with open source licenses. The Legal Counsel focuses on four essential duties:

1. Provide approval for the use of open source in products/services

The approval of the Legal Counsel is required when using open source in a commercial product. Typically, the Legal Counsel reviews the compliance ticket in the online tracking system (for instance, JIRA or Bugzilla), the resulting report from the scanning tool, and the license information provided with the source code package. They then evaluate risk factors based on feedback provided by both engineering and the open source compliance officer.

As part of this exercise, the Legal Counsel decides on incoming and outgoing licenses of the software component in question. The incoming licenses are the licenses of all source code included in a given body of code; the outgoing licenses are the licenses under which the source code and/or object files are being made available to its recipients.

2. **Advise on open source licensing**

    - Offer guidance about open source license obligations.
    - Advise on licensing conflicts: Such conflicts arise when combining code licensed under incompatible or conflicting licensing terms.
    - Advise on IP issues associated with the use of open source. This is especially the case when the company is about to release proprietary source code under an open source license(s).
    - Provide recommendations and guidance on open source questions and concerns.

3. **Review and approve updates to end-user documentation**

This form of legal support is related to ensuring that appropriate open source notices (copyright, attributions, and license notices) are provided to consumers in relation to any open source included in the product. In addition, if there is source code licensed under one of the GPL/LGPL family of licenses, a written offer needs to be provided along with information on how to obtain the source code.

4. **Contribute to establishing and managing the compliance program**

    - Establish and maintain the open source policy and process.
    - Handle inquiries sent to the company in relation to open source compliance.
    - Provide training around open source licenses, company policies, and guidelines.

ENGINEERING AND PRODUCT TEAMS

Engineering and product teams may have one or more representatives who participates in the OSRB, track down all compliance-related tasks assigned to engineering, and ensure proper resolution.

In parallel, engineering and product teams have several responsibilities with respect to open source compliance:

- Submit requests to use open source software: Engineering and product teams decide what external software to bring into the product, including third party and open source. Their primary responsibility from a compliance perspective is to submit a usage form for any open source planned for inclusion in a product or service. The form describes the intended use of the open source in question and helps construct and maintain a good record of software origin and provenance.

- Follow technical compliance guidelines: Engineering and product teams should follow OSRB technical guidelines to architect, design, integrate, and implement source code. The OSRB guidelines typically cover:

    - Common mistakes and how to avoid them

    - Rules to follow when integrating libraries and other middleware to avoid linkage issues that might arise

    - Development in kernel space versus user space (on Linux), especially with whole-platform development in embedded environments

    - Specific engineering situations that are applicable to open source compliance

- Conduct design reviews: Engineering teams should continuously conduct design reviews to discover and remedy any compliance issues in a timely manner.

- Cooperate with OSRB: Engineering teams must respond promptly to questions asked by the OSRB and cooperate in resolving compliance tickets.

- Track source code changes: Engineering should maintain a change log for each modified source code. Depending on the open source license in questions, some licenses (such as the GPL/LGPL family of licenses) mandate that modified files to carry prominent notices stating that you changed the files and the date of the change(s).

- Prepare source code packages for distribution: Engineering teams prepare the source code packages for publication as part of

meeting open source license obligations. Chapter 5 discusses other source code distribution methods.

- Integrate compliance milestones as part of the development process: This exercise takes place in collaboration with the OSRB and the Compliance Officer.
- Undergo open source training: All engineers must take the available open source compliance training.
- Monitor the open source projects to determine whether any bug fixes or security patches have become available, and take responsibility for updating the open source component used in the product. The individual package owner within the organization usually performs this specific task.

COMPLIANCE OFFICER

The Compliance Officer, Manager/Director of Open Source, chairs the OSRB and manages the compliance program. This role is typically accountable for the following core responsibilities with respect to open source compliance:

- Managing and executing company-wide Open Source strategy and business metrics that track business and technical success of the program
- Lead cross-functional OSRB that acts as a clearinghouse for all inbound and outbound open source software activities
- Conceives, implements and executes internal and external processes that allow the company to be an aggressive consumer of open source software and a participant in open source communities

The compliance officer must possess as many as possible of the following skills:

- Proven record of accomplishment creating and executing an open source strategy and tactics for a commercial ISV or IHV
- Strong teamwork skills that demonstrate the ability to drive cross-functional alignment across engineering, marketing and business development disciplines

- Strong technical / engineering background to engage directly with engineering teams, development partners, and industry consortia to assess and drive opportunities
- Success conceiving and deploying lightweight internal and external processes cross-functionally that drive business success
- Strong written and verbal communications skills
- Demonstrated ability to act as the primary internal and external evangelist for open source
- Strong existing relationships with relevant open source communities, industry consortia, and open source foundations
- Solid understanding of common open source licenses to discuss with legal counsel
- Knowledge of industry practices
- Knowledge and experience in establishing corporate-wide policies and processes

In addition to the responsibilities pertaining to the OSRB, the Compliance Officer carries the following duties:

- Drive the compliance process and act as the compliance program manager, ensuring all compliance-related tasks are addressed and there are no compliance issues blocking products from shipping
- Coordinate source code scans and drive all auditing issues to closure
- Participate in engineering design reviews, code inspections, and distribution readiness assessments to assure that the engineering and product teams follow all compliance processes and policies and conform to the approved OSRB usage form
- Coordinate source code distribution of open source packages (when stipulated by licenses) with engineering and product teams, including preparing and verifying a distribution checklist for each open source package

- Act as liaison between OSEC and OSRB
- Escalate compliance issues to OSEC
- Act as liaison between the engineering and product team and the OSRB and OSEC in regard to usage plan approval processes
- Report on compliance activities to the OSEC, including flagging issues that prevent shipping a product or service

OPEN SOURCE EXECUTIVE COMMITTEE

The Open Source Executive Committee (OSEC) consists of engineering, legal, and product marketing executives in addition to the Compliance Officer. The OSEC is responsible for setting open source strategy, reviewing and approving release of intellectual property, and launching new open source projects.

DOCUMENTATION

The documentation team is responsible for incorporating written offer and any appropriate open source notices in the product documentation. Figure 9 provides the basic workflow of how such notices are prepared, approved, and added to the product manual or other form of documentation.

Compliance Officer	Prepares the draft notices document based on the final software bill of material in addition to a proposal on where and how these notices should be presented.
Legal Counsel	Reviews, edits and approves proposal from the Compliance Officer.
Documentation Team	Update product documentation as approved by Legal.

Figure 9. The role of the documentation team in updating the product documentation, reflecting the presence of open source in the product

The process starts with the compliance officer preparing the draft of the written offer, license, copyright and attributions notices for all open

source software in the product (or software stack). Next, the legal counsel reviews the draft proposed by the compliance officer, provides modification proposals if needed, and pushes the final version to the documentation team. The last step of the process is including the final text in the product documentation.

LOCALIZATION

The localization team is responsible for translating basic language that informs users of the availability of open source software in the product and directs them to the proper notices made available in English.

SUPPLY CHAIN

Supply chain (software procurement) procedures must be updated to address the acquisition and use of open source. It is highly recommended that you examine software supplied to you by third party software providers.

Supply chain personnel are usually involved in moving software from the suppliers to your company. Supply chain can support open source compliance activities by mandating that third party software (and hardware) providers disclose any open source that is being delivered with their wares, and by assisting with licensing-in third party software that is bundled with and/or integrates open source packages.

A best practice in this area is to mandate that third party software providers disclose any open source used in their offering, along with a statement on how they plan to meet the applicable open source license obligations. If third party software includes open source, supply chain must ensure that open source license obligations are satisfied, since, after initial ingress, those obligations become your responsibility as distributor of a product or service that includes open source. It is not acceptable to point “upstream” to a supplier and to inform recipients of your code that meeting license obligations was the responsibility of the supplier instead of your own.

IT

IT provides support and maintenance for the tools and automation infrastructure employed by a compliance program. This responsibility spans the servers hosting the various tools, the tools, mailing lists, and web portals. In addition, IT may receive requests from the OSRB to develop and/or acquire tools that will be used to improve effectiveness and automation of the compliance activities.

CORPORATE DEVELOPMENT

Corporate Development is involved with open source compliance in two major scenarios: mergers and acquisitions transactions, and outsourced development.

Mergers and Acquisitions

If a company is considering a merger or is the target of an acquisition, it should structure its compliance program to offer a level of disclosure and provide representations. Company policies regarding merger and acquisition transactions need to be updated to account for open source. Corporate Development must mandate that source code be evaluated from a compliance perspective prior to any merger or acquisition to avoid surprises that might derail discussions or affect the company's valuation. For the acquiring company, comprehensive code evaluation assures accurate valuation of software assets and mitigates the risk of unanticipated licensing issues undermining future value. In addition, the acquiring company may include provisions in the purchase agreement requiring the disclosure of open source that is subject to the transaction.

Diligence practices should be updated to require open source disclosure and include guidance regarding the review of any disclosed open source and licenses.

Chapter 13 is dedicated to the discussion of open source audits that occur as part of an M&A transaction.

Outsourced Development

Agreements relating to outsourced development of software should also be updated to reflect compliance procedures and to ensure that other

provisions of these agreements (such as representations and warranties) are broad enough to cover any potential risks posed by specific use case of incorporating open source. Corporate Development must mandate that all source code received from outsourced development centers must go through the compliance process to discover all open source being used and to ensure proper actions to fulfill license obligations.

Other Corporate Transactions

Corporate Development is also involved with compliance in transactions such as spin-offs and joint ventures. In some cases, the compliance due diligence may result in a decision not to proceed with the transaction, if that the compliance situation proves far from ideal or even risky if the other parties have poor compliance practices.

Chapter 4

OPEN SOURCE COMPLIANCE PROCESS

Implementation of open source compliance processes can vary across organizations based on a number of factors including the underlying development processes into which compliance must fit, the size and nature of the code base, the number of products or services involved, the amount of externally supplied code, the size and organizational structure, and many more. However, the core elements of compliance usually remain the same: identifying the open source in the code base, reviewing and approving its use, and satisfying obligations.

This chapter focuses on the core elements of a compliance process. The result of compliance due diligence is identification of all free and open source software used in a product intended for external distribution, and a plan to meet the attendant license obligations. Figure 10 offers a high-level overview of a sample end-to-end compliance process.

Figure 10. Simplified view of the compliance end-to-end process

Throughout this chapter, we discuss the phases of a compliance process, their inputs and outputs. In addition, we examine how we can manage software usage via a compliance process.

EFFECTIVE COMPLIANCE

The term due diligence refers to a number of concepts involving source code inspection, source code surveillance, or the performance of quality duties and system audits. In the case of open source compliance, due diligence is required to ensure the following:

- Open source software used in the product has been identified, reviewed, and approved
- Product implementation includes only approved open source components and licenses.
- All obligations related to the use of licensed material have been identified
- Appropriate notices have been provided in documentation, including attributions and copyright notices
- Source code, including modifications (when applicable), has been prepared and is available at the time the product ships
- Verification of all the steps in the process to ensure correctness

There are great benefits to having an end-to-end compliance process that is simple and well understood within the organization. Such a process would:

- Enable organizations to benefit from open source while complying with obligations
- Move open source use from ad hoc to a standardized process that is well understood and followed
- Help manage acquisition of open source code via third party providers
- Help employees understand how to work with open source in a responsible way
- Improve the relationship with developers in the various open source projects used by your organization

- Accelerate exchange of information and ideas with the project communities of integrated code through upstreaming any source code modifications
- Speed innovation, since the organization is able to safely adopt open source components and use them as enablers for new services and products

ELEMENTS OF AN END-TO-END COMPLIANCE PROCESS

Figure 11 (next page) illustrates the ten key steps in an end-to-end compliance process:

1. Identification of incoming source code
2. Auditing source code
3. Resolving any issues uncovered by the audit
4. Completing appropriate reviews
5. Receiving approval to use open source
6. Registering open source in the software inventory
7. Updating product documentation to reflect open source usage
8. Performing verification of all previous steps prior to distribution
9. Distributing source code packages
10. Performing final verifications in relation to distribution

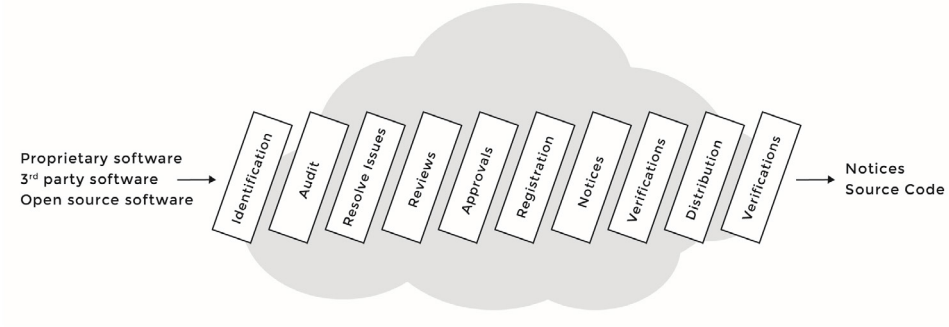


Figure 11. End-to-end compliance process

The remainder of this chapter will address each of these ten steps in detail.

Step 1 – Identification of Open Source

The goal of this initial step is to monitor the ingress and incorporation of open source in a software portfolio, regardless of whether that incorporation was as a standalone package or embedded within third party or company-developed software. There are several methods to identify open source used in the product:

- A request to use open source: This is the most common method for identifying the usage of open source in a product. Engineering staff or Product Management staff are required to inform the Open Source Review Board (OSRB; described in Chapter 3) or the compliance team of their intent to use open source in a specific product or platform release. The submitter provides information regarding the intended use of the open source package for review and approval.
- Auditing the full platform or product code base to establish a compliance baseline, and then auditing code modules that have changed in subsequent releases or those that have entered the code base.
- Third party due diligence: This involves requiring a full disclosure of open source components and snippets provided by third party

suppliers, with an accompanying review of the disclosure by the open source compliance team. In some cases, it will make sense to require third party software vendors to provide an audit of the supplied code as an additional layer of diligence. This helps ensure you are controlling the intake of open source.

- Auditing proprietary (company-developed) software components: In some instances, engineers may decide to copy/paste source code from open source components and include it in proprietary software components. Therefore, it is important to audit company-developed software components, since they may include open source code, which may lead to compliance failures if not discovered before product ship date.

- Inspect all open source components and snippets entering the organization source code repository that do not correspond to an incoming request to use open source: Relying on engineers to fill out forms announcing their intent to use open source is not always a reliable method to account for all incoming open source software. Therefore, as a backup, we recommend setting up a source code control system with a separate folder for open source and a notification alert any time there is a check-in to this master folder. Since it is always a recommended practice to separate open source, company-developed proprietary software, and third party software in different folders in your build system, it becomes feasible to set up alerts when new code is being submitted. If a submitted does not correspond to an existing usage form (open source request form), then it is a new component (or a newly introduced snippet) and requires that a new form be filled out.

Identification phase prerequisite

One of the following conditions is met:

- An incoming OSRB form requesting using a specific open source

- Discovery of open source code used (without proper authorization) via a complete platform scan

- Discovery of an open source being used as part of third party software

Identification phase outcome

- A compliance record is created (or updated) for the open source
- An audit is requested to scan the source code

Step 2 – Auditing Source Code

The second step in compliance due diligence consists of scanning the source code using automated analysis tools to discover matches with known open source projects. The auditing personnel perform a source code scan iteratively from one release to another. The goal of this exercise is to build a chain of evidence that proves what is included in the release is compliant with the various applicable open source licenses.

- The goals of the audit are to:
- Update the release bill of materials to account for any open source added (or removed) since the last previous scan
- Confirm the origin(s) of the source code, including the provenance of any open source
- Flag any dependencies, code matches, and licensing conflicts

Auditing phase prerequisite

A compliance record (also called a ticket) is created capturing all information about the usage of that specific open source and providing the location of the source code within the internal build system. In some cases, specifically when a full platform scan is done, an open source component may be scanned before having a proper compliance report. In this case, a record is created when the component is discovered.

Auditing phase outcome

- An audit report identifying the origins and licenses of the scanned source code
- Change request tickets are filed against the appropriate engineering team for any issues identified during the audit that require engineering rework

Figure 12 illustrates the actions that can trigger a software audit.

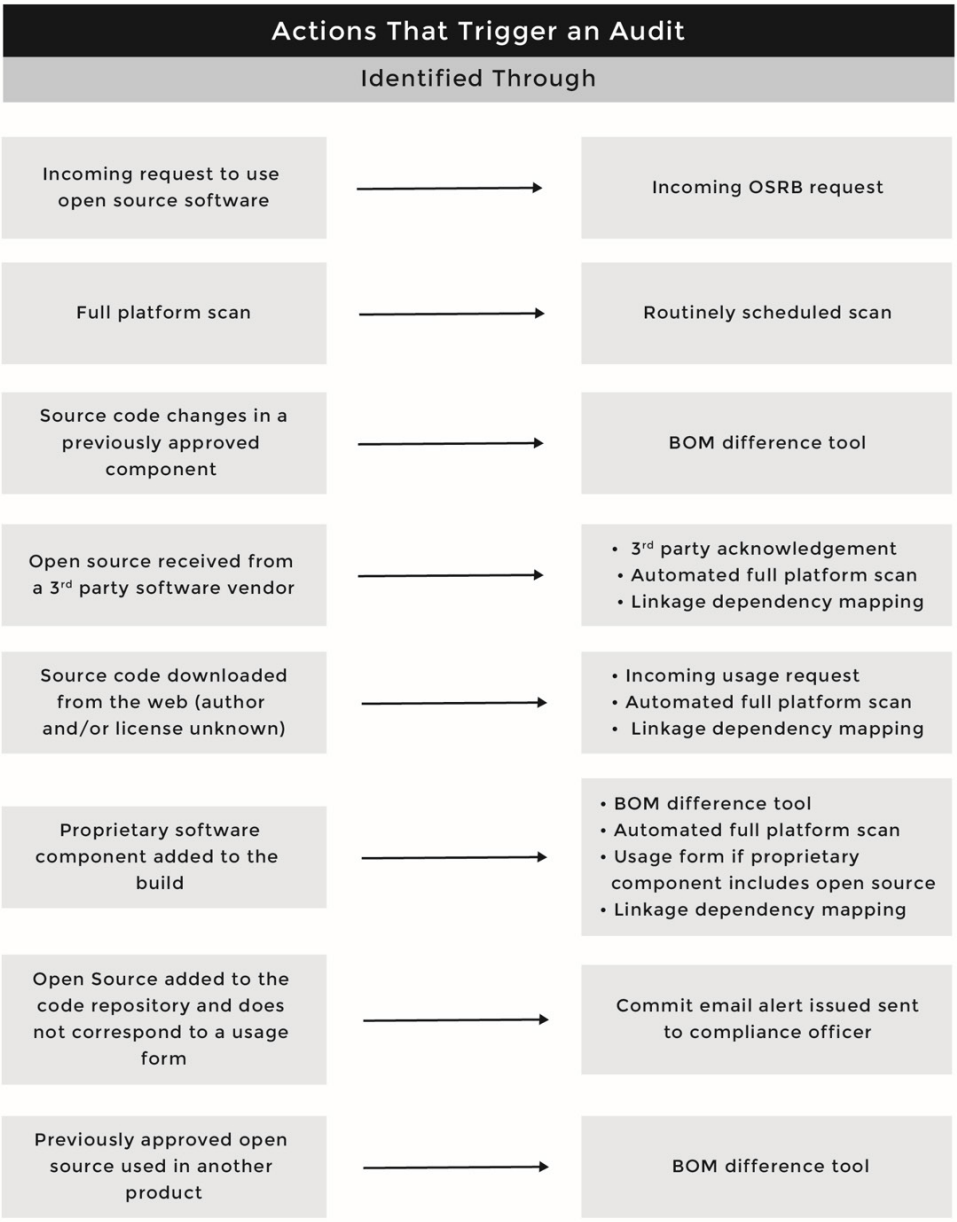


Figure 12. Triggers leading to the identification and audit request for incoming new code

Step 3 – Resolving Issues

In this step of the process, all issues identified during the auditing step are resolved. The OSRB Chair monitors closure of tickets assigned to engineers during the Audit step. Once the engineers have resolved the identified issues, the OSRB Chair should request a new audit to confirm that the resolved issues no longer exist.

Resolving Issues phase prerequisite

A source code scan has been completed, and an audit report is generated identifying the origins and licenses of the source code. The report has flagged source code files that were not identified, or possible license conflicts resulting from mixing source code incoming under different licenses. The Compliance Officer will drive the effort to resolve these issues.

Resolving Issues phase outcome

A resolution for each of the flagged files in the report and a resolution for any flagged license conflict.

Step 4 – Reviews

Once the auditing is complete and all issues identified earlier have been resolved, the compliance ticket for a specific software component or snippet moves in the process to the review phase. Figure 13 (next page) illustrates the various reviews performed on a given compliance ticket. The reviewers need to understand the licenses that govern use, modification, and distribution of the source code in question, and identify the obligations of the various licenses. For any given software component or snippet, the reviewers of the compliance ticket are:

- Internal package owner (the developer working on specific source code component)
- Source code scanning or auditing personnel
- OSRB (Open Source Review Board), which includes OSRB chair

(Compliance Officer), Legal counsel, and OSRB engineering representative

- OSEC (Open Source Executive Committee) in the event the company will open source proprietary source code.

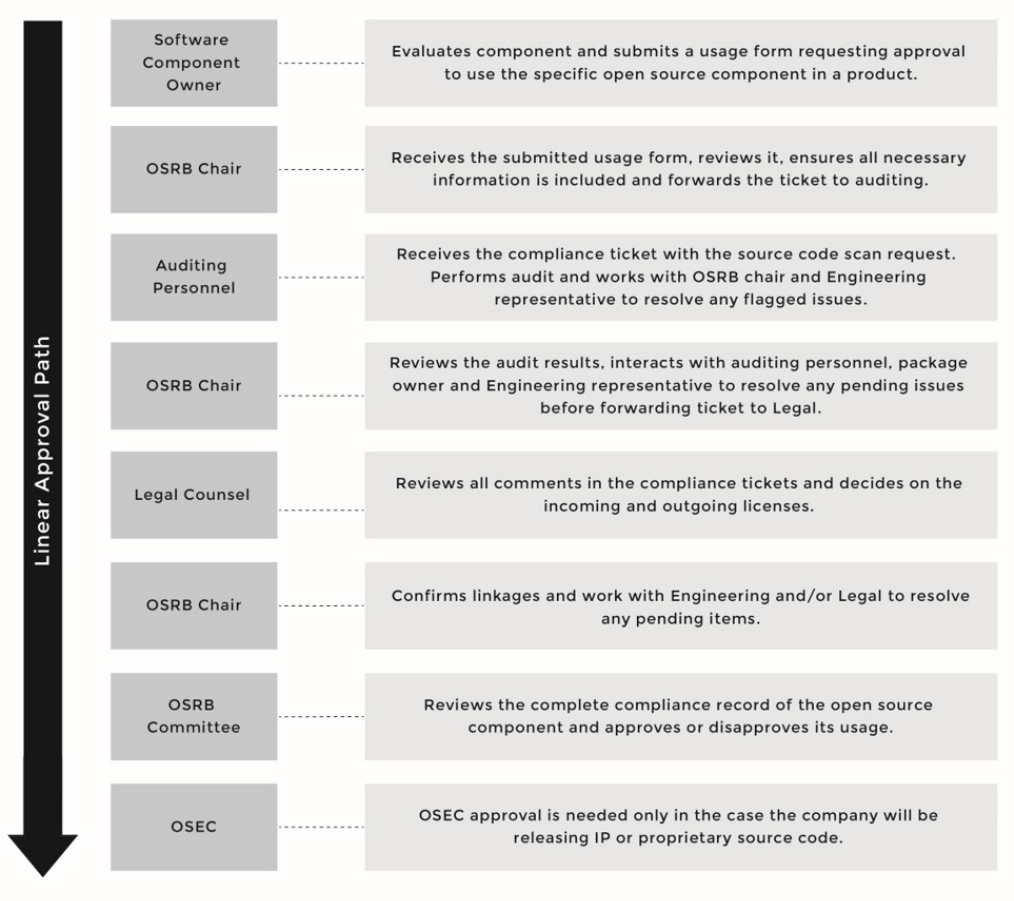


Figure 13. Reviewers of the compliance ticket and their roles

As part of this step of the compliance process, there are two important reviews: the architecture review and the linkage analysis review.

Architecture Review

The goal of the architecture review is to analyze any interactions between the open source, third party, and proprietary code. The result of the architecture review is an analysis of the licensing obligations that may extend from the open source components to the proprietary components (and vice versa). The internal package owner, the OSRB engineering representative, and the open source expert usually perform the architecture review. If they identify a dependency resulting in a licensing conflict, the OSRB Chair will issue a ticket to Engineering to resolve the identified problem by reworking the source code..

Linkage Analysis Review

The goal of a linkage analysis review is to find potentially problematic code combinations at the static and dynamic link level, such as linking a GPL or more commonly an LGPL-licensed library to a proprietary source code component. The OSRB Chair performs this review using an automated tool. Similarly to the architectural review, any linkage conflicts are reported to Engineering to resolve.

Review phase prerequisite

Source code has been audited and all issues have been resolved.

Review phase outcome

OSRB members perform an architecture review and a linkage analysis for the specific component and mark it as ready for approval.

Step 5 – Approvals

Once reviews are completed, the compliance ticket moves to the approval step, where the OSRB will either approve it for usage or reject it with explanation and a possible proposal on how to fix that. For most software components, the OSRB grants approval once a ticket has progressed that far in the process.

Once the OSRB approves the usage of an open source component, the OSRB communicates the approval to the product teams so they understand their responsibilities and begin preparations to fulfill the license obligations. If the OSRB rejects the use of the open source component, they communicate the reason for rejection to the requester, and this information is recorded as part of the compliance ticket. As a result, the open source component cannot be used in the product, although the requester can consider submitting an appeal for reconsideration by the OSRB.

Approvals phase prerequisite

All OSRB members have reviewed the compliance ticket, and the OSRB has completed the architecture review and linkage analysis.

Approvals phase outcome

Approve or reject the use of the specific component or snippet with explanation of the reason and possibly a proposal to reverse the decision when it's a rejection.

Step 6 – Registration

Once the OSRB approves a software component for use in a product or as part of a service, its compliance ticket will be updated to reflect the approval. The component (or in some case snippets, or part of a component) is added to the software inventory that tracks open source use and use cases.

Companies that follow a conservative approach with their compliance practices, you would approve open source software for a specific version and usage in a specific product or service version. If a new version of this open source software becomes available, companies usually require a new approval to ensure that the usage model and even the license are still in line with the internal policy.

Registration phase prerequisite

The OSRB has approved the component's usage in the product.

Registration phase outcome

The component is registered in the software inventory, with the component name, version, internal owner, and the details of where the component is being used, such as product name, version, release number, etc.

Step 7 – Notices

One of the key obligations when using open source is the documentation obligation also referred to as the notice obligation. Companies using open source in an externally distributed product or service must:

- Inform the end user how to obtain a copy of the source code that's been made available as a result of meeting the license obligations (when applicable)
- Acknowledge the use of open source by providing required copyright, attribution, and license notices for all applicable open source software (components and snippets)
- Reproduce the entire text of the license agreements for the open source code included in the product

If companies are non-compliant with open source license obligations, they are exposed to possible legal action by the copyright holder for copyright infringement, and can potentially lose the right to use and distribute the software in question. In order to fulfill documentation obligations, appropriate notices must be included with the product. In this step of the compliance process, the OSRB Chair prepares the notices and passes them to the appropriate departments for fulfillment.

Notice phase prerequisite

The OSRB has approved the use of the open source code and registered in the software inventory.

Notice phase outcome

The license, copyright, and attribution notices for are prepared and passed to the appropriate departments to be included in the product documentation.

Step 8 – Pre-Distribution Verifications

The next step in the compliance process is to decide on the method and mode of distribution, type of packages to distribute, and mechanism of distribution.

- The goals of the pre-distribution verification are to ensure that:
- Open source packages destined for distribution in fulfillment of license obligations have been identified and approved
- Source code packages (including modifications) have been verified to match the binary equivalent shipping in the product
- Appropriate notices have been included in the product documentation to inform end-users of their right to request source code for identified open source
- All source code comments have been reviewed and any offending or inappropriate content has been removed. This is not strictly a compliance issue; however, in some cases, an innocent comment about where the code was received can trigger a larger compliance question.

Pre-Distribution Verifications phase prerequisites

- The OSRB has approved the component or snippet for usage.
- It has been registered in the software inventory.
- All notices have been captured and sent for fulfillment.

Pre-Distribution Verifications phase outcome

- Decide on distribution method and mode.
- Ensure that all the pre-distribution verifications have been completed.

Step 9 – Distribution

Once all pre-distribution verifications have been completed, it is time to upload the open source packages to the distribution website.

Packages will be labeled with the product and version it corresponds to (this scenario assumes that you have chosen this method to make source code available; other methods are discussed in a later chapter). Note that this action is helpful to those desiring code download but may not be sufficient by itself to satisfy license obligations. Furthermore, a recommended practice is to provide email and postal mail contact information for any compliance or open source-related questions.

Distribution phase prerequisite

All pre-distribution verifications have been checked and no issue discovered.

Distribution phase outcome

The source code of the component in question is uploaded to the website for distribution (if that is the distribution method of choice).

Step 10 – Final Verifications

Once you upload the open source packages to the distribution website, validate that the packages have been uploaded correctly and can be downloaded and uncompressed on an external computer without errors. If you are providing a patch, ensure that it applies cleanly and that you have specified the proper version of the upstream component.

Final verifications phase prerequisite:

The source code is published on the website.

Final verifications outcome

You receive verification that the source code is uploaded correctly and accessible for download, and that it corresponds to the same version that was approved.

Chapter 5

COMPLIANCE PROCESSES AND POLICIES

For the purpose of this book, the focus of the discussion is using and integrating open source with proprietary and third party source code in a commercial product. The discussions exclude policies and processes for using open source solely inside your organization for testing and evaluation purposes. This chapter discusses usage policy and process in addition to the base and incremental compliance process, and guidelines for achieving incremental compliance.

POLICY

The usage policy is an essential building block in a compliance program. This policy does not have to be lengthy or complicated. A simple policy can be effective as long as it mandates the following basic rules or principles:

- Engineers must receive approval from OSRB before integrating any open source in a product.
- All software must be audited and reviewed, including proprietary software components, software received from third party providers, and open source software, to ensure license obligations can be fulfilled before product ships.
- Product must fulfill open source licensing obligations prior to customer receipt.
- Approval to use open source code for one product is not a blanket approval to use that code in every possible context. Approvals are given on a case by case. If engineers want to re-use the open source code for another deployment, a new approval is required.
- All changed open source components and snippets must go through the approval process. This requirement arise from the fact that some projects change licenses with new releases and

downstream users of the code need to be aware of such changes and not assume the same license is still in effect.

These rules ensure that any software (proprietary, third party, open source) that makes its way into the product base has been audited, reviewed, and approved. Furthermore, it ensures that the company has a plan to fulfill the license obligations resulting from using software originating from multiple sources.

PROCESS

The compliance usage process includes scanning the source code of the software package in question, identifying and resolving any discovered issues, performing legal and architectural reviews, and making a decision regarding the usage approval for a given software package.

Figure 14 illustrates a simplistic view of a compliance usage process. This figure does not demonstrate the iterative nature of such a process; a more elaborate view is provided in Figure 17 (page 89).

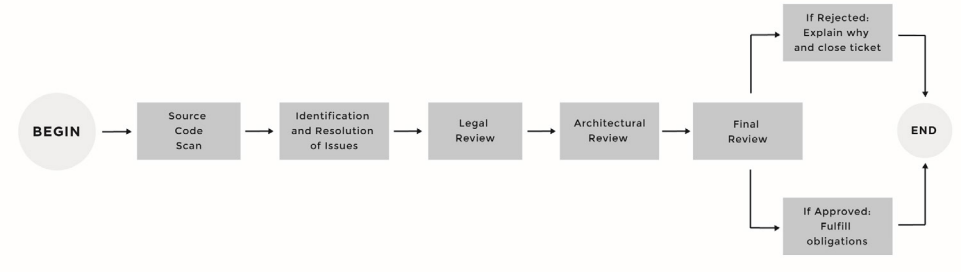


Figure 14. A sample compliance usage process

Source Code Scan

In the scanning phase, all the source code is scanned using a source code scanning tool. Chapter 13 offers a detailed discussion on the various metrics to use and look for when evaluating such tools.

Figure 15 illustrates some of the factors that can trigger a source code scan.

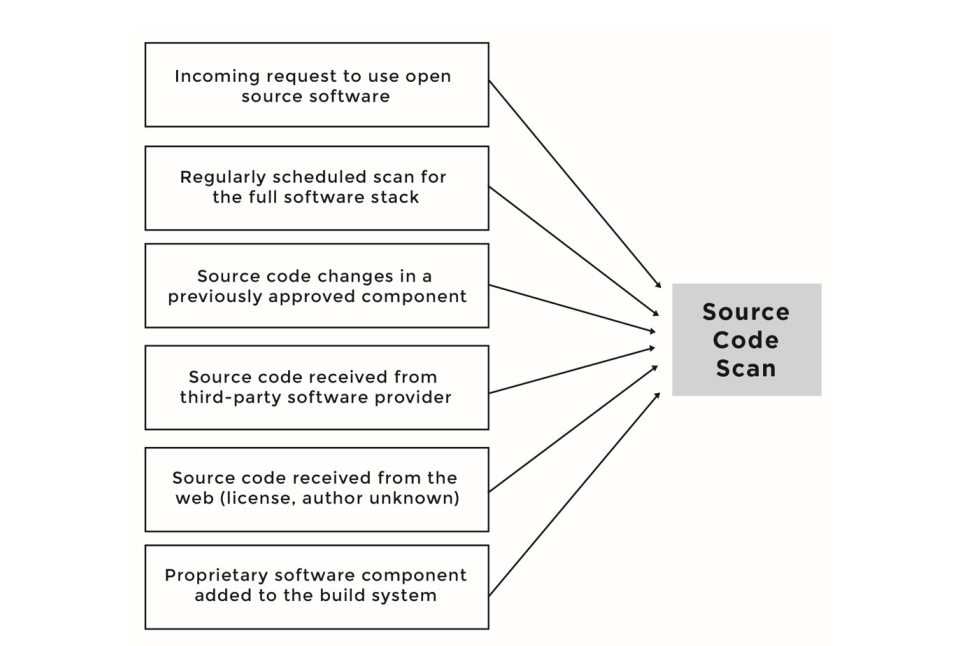


Figure 15. Events that trigger a source code scan

The triggers include:

- An incoming usage form from engineering staff: A developer fills out the OSRB usage form to provide basic information about the source code in question they intend to use. Upon the submission of the form, a compliance ticket (in a system such as JIRA or Bugzilla) will be created automatically, and a source code scan request will be sent to the auditing staff.
- A periodically scheduled full-platform scan: Such scans are very useful to uncover open source that may have snuck into your software platform without an OSRB form.
- Changes in previously approved software components: In many cases, engineers start evaluating and testing with a certain version of an OSS component, and later adopt that component when a new version is available.

- Source code received from a third party software provider who may or may not have disclosed open source.
- Source code downloaded from the web with unknown author and/or license, which may or may not have incorporated open source in it.
- A new proprietary software component entering the build system where engineering may or may not have incorporated open source into it.

In preparation for legal review, the individual who runs the source code scan should attach to the compliance ticket all the license information found in the package, such as any COPYING, README, or LICENSE files, in addition to any files stating copyright and attribution notices.

Identification and Resolution

In the identification and resolution phase, the auditing team inspects and resolves each file or snippet flagged by the scanning tool.

Legal Review

In the legal review phase, the legal counsel reviews reports generated by the scanning tool, the license information of the software component, and any comments left in the compliance ticket by engineers and members of the OSRB. If there were no issues with the licensing, the legal counsel would then decide on the incoming and outgoing licenses of the software component, and would forward the compliance ticket into the compliance architectural phase. If a licensing issue is found, for example mixed source code with incompatible licenses, the legal counsel will flag these issues and reassign the compliance ticket to Engineering to rework the code. In some cases, if the licensing information is not clear or if it is not available, the legal counsel contacts the project maintainer or the open source developer to clarify the ambiguities and to receive a confirmation of the license under which that specific software component is licensed.

Architecture Review

In the architecture review, the compliance officer and the engineering OSRB representative perform an analysis of the interaction between the open source, proprietary, and third party code.

This is accomplished by examining an architectural diagram that identifies:

- Open source components (used “as is” or modified)
- Proprietary components
- Components originating from third party software providers
- Component dependencies
- Communication protocols
- Other open source packages that the specific software component interacts with or depends on, especially if they are governed by a different open source license

The result of the architecture review is an analysis of the licensing obligations that may extend from open source to proprietary or third party software components (and across open source components as well). If the compliance officer discovers any issues, such as proprietary software component linking to a GPL licensed component, the compliance officer forwards the compliance ticket to Engineering for resolution. If there are no issues, then the compliance officer moves the ticket to the final stage in the approval process.

Final Review

The final review is usually an OSRB face-to-face meeting during which the OSRB approves or denies usage. In most cases, if a software component reaches the final review, it will be approved unless a condition has presented itself (such as the software component no longer being in use). Once approved, the compliance officer will prepare the list of license obligations for the approved software component and pass it to appropriate departments for fulfillment.

PROCESS STAGES' INPUTS AND OUTPUTS

In this section, we discuss the inputs and outputs of each of the five phases in the OSRB usage process, as illustrated in Figure 16. Please note that these phases are for illustration purposes and may not be the same as the steps in your specific scenario.

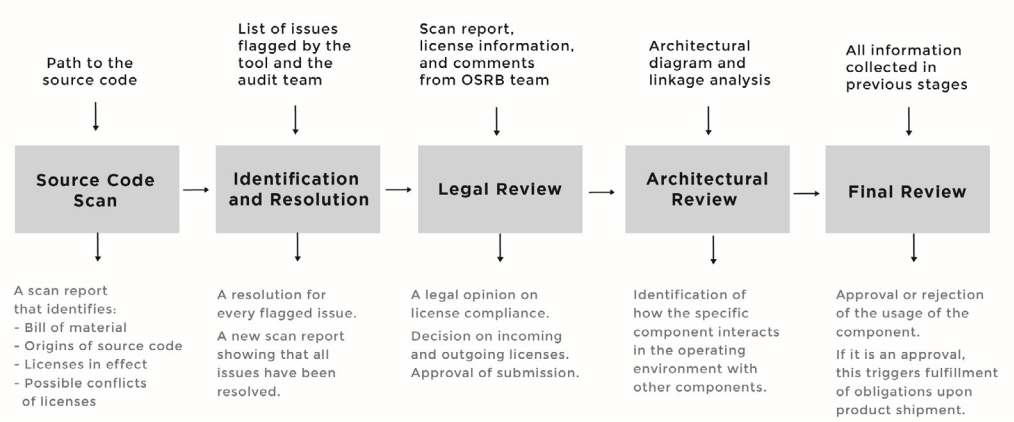


Figure 16. Inputs and outputs of the usage process

Source Code Scan Phase

Input

The input to the scan phase is the usage form that an engineer has filled out online and submitted. Table 6 (on page 99) provides details on the form. It includes all information about the open source component in question, in addition to the location of the source code. Periodic full-platform scans should also take place every few weeks to ensure that no open source software component or snippet has been included into the code base without a corresponding OSRB form.

Output

The output from the scan phase is a report produced by the source code scanning tool.

- The report provides information on:
- Known software components and open source snippets in use, also better described as the software Bill of Materials (BoM)
- Licenses in effect, license texts and summary of obligations
- License conflicts (to be verified by legal) as flagged by the tool
- File inventory
- Identified files
- Dependencies
- Code matches
- Files and source code matcher pending identification

Identification and Resolution Phase

Input

The input to this phase is the report generated by the scanning tool in the previous phase. The report flags issues such as conflicting and incompatible licenses. If there are no issues, then the compliance officer will move the compliance ticket forward to the legal review phase. If there are issues to be resolved, then the compliance officer creates subtasks within the compliance tickets and assigns them to the appropriate engineers to be resolved. In some cases, a code rework is needed; in other cases, it may simply be a matter of clarification. The subtasks should include a description of the issue, a proposed solution to be implemented by engineering, and a specific timeline for completion.

Output

The output is the resolution of all issues. The compliance officer may request a new scan of the code to confirm that earlier issues have been resolved. The compliance officer then forwards the compliance ticket to the representative from the Legal department for review and approval.

Legal Review Phase

Input

When a compliance ticket reaches the legal review phase, it already contains:

- A scan report is attached to the compliance ticket.
- A confirmation from engineering and the compliance officer that all discovered issues during the source code scan has been resolved.
- Copies of the license information attached to the ticket: Typically, the compliance officer attaches the README, COPYING, and AUTHORS files available in the source code packages to the compliance ticket. Other than the license information, which for open source software components are usually available in a

COPYING or a LICENSE file, you need to capture copyright and attribution notices as well. This information will provide appropriate attributions in your product documentation.

- Feedback from the compliance officer regarding the compliance ticket (concerns, additional questions, etc.).
- Feedback from the engineering representative in the OSRB or from the engineer (package owner) who follows/maintains this package internally.

Output

The output of this phase is a legal opinion, and a decision on the incoming and outgoing license(s) for the software component in question. The incoming and outgoing licenses are in the plural form because in some cases, a software component can include source code incoming from multiple source and available under different licenses.

Incoming and outgoing licenses

The incoming license is the license under which the company receives the software component or snippet. The outgoing license is the license under which the company is licensing (or relicensing) the software component or snippet.

In some cases, when the incoming license is a permissive license that allows relicensing, companies will relicense that software under their own proprietary license.

A more complex example would be a software component that includes proprietary source code, source code licensed under License-A, source code that is available under License-B, and source code available under License-C. During legal review, the legal counsel will need to decide on the incoming and outgoing license(s):

Incoming licenses = Proprietary License + License A + License B + License C
Outgoing license(s) = ?

Architecture Review Phase

The goal of the architecture review is to analyze the interactions between the open source code and third party and proprietary code. The result of the review is an analysis of the licensing obligations that may extend from the open source components to the proprietary components. The internal package owner, the OSRB engineering representative, and the Compliance Officer usually perform the architecture review. If they identify a licensing conflict, the Compliance Officer issues a ticket to Engineering to fix the issue.

Input

Source code has been audited and all issues have been resolved.

Output

OSRB members perform an architecture review for the specific component, and mark it as ready for the next step (i.e., Final Approval) if no issues were uncovered during the architecture review.

Final Approval Phase

Input

The input to this phase is the complete compliance record of the software component, which includes the following:

- A source code scan report generated by the scanning tool.
- The list of discovered issues, information on how they were resolved, and who verified that these issues were successfully resolved.
- Architectural diagrams and information on how this software component interacts with other software components.
- Legal opinion on compliance, and decision on incoming and outgoing licenses.

- Dynamic and static linkage analysis, if applicable in an embedded environment (C/C++).

Output

The output of this phase is a decision to either approve or deny the usage of the software component.

DETAILED USAGE PROCESS

Many possible circumstances can affect compliance procedures. Figure 17 (next page) provides a detailed process that highlights several possible scenarios, and how to move from one step to another in the process. We then discuss eight possible scenarios. These scenarios are not mutually exclusive and are not the only possible scenarios; however, we use them for illustration and discussion purposes.

The scenarios illustrated in Figure 17 (next page) are the following:

Scenario 1: The source code is 100% proprietary. The audit team did not find any open source code.

Scenario 2: The source code includes open source code from multiple sources with incompatible licenses.

Scenario 3: Linkage issue were identified during architectural review.

Scenario 4: A source code package is not used anymore.

Scenario 5: Due diligence identified IP that will be released to meet license obligations.

Scenario 6: An issue was identified during the verification phase that needs to be resolved.

Scenario 7: Source code is approved for use.

Scenario 8: Source code is rejected.

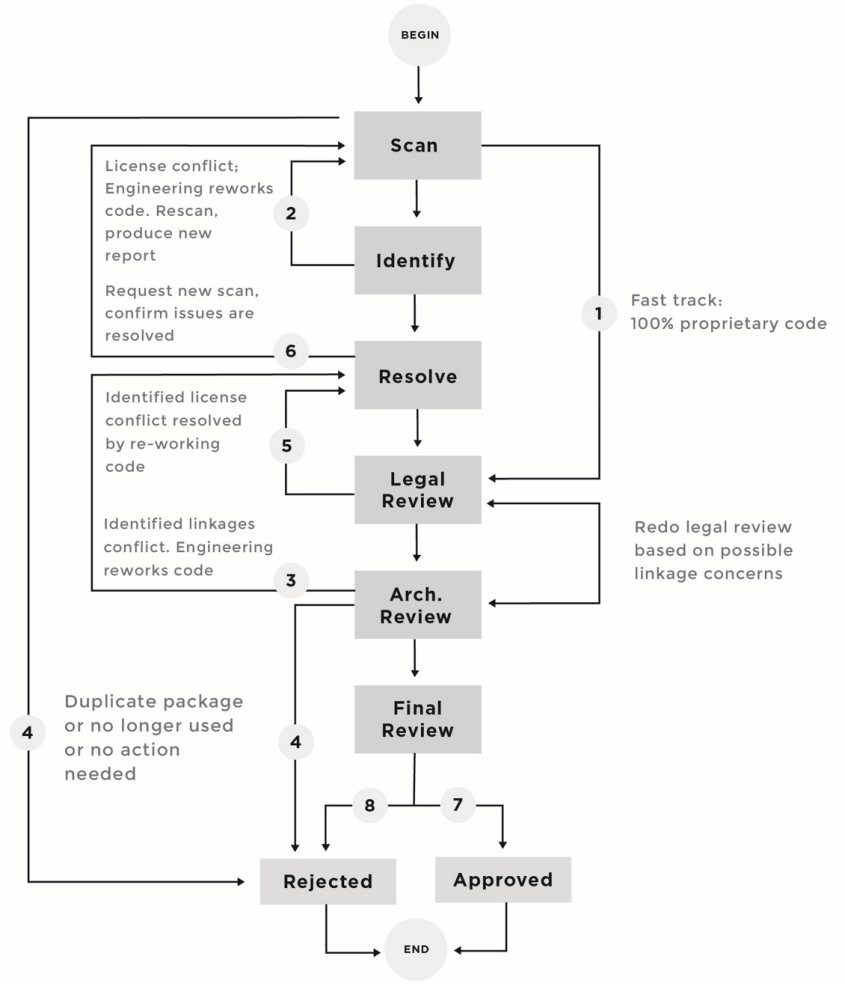


Figure 17. Specific compliance scenarios

Scenario 1: Source code is 100% proprietary

The scanned software component is 100% proprietary code, and no open source code is declared or identified. In this case, we assume the fast track, and the compliance ticket for that specific component will be forwarded for legal review. The legal counsel decides on the license to attach to this proprietary component, and forwards it to the compliance officer to perform architectural and linkage analysis.

Scenario 2: Incompatible licenses

The scanned software component includes source code that originated from multiple sources with incompatible licenses. Another example would be a software component with a mix of proprietary source code and source code licensed under the GPL. In this scenario, the scan report is attached to the compliance ticket and assigned to the developers with a request to rework the code by removing the GPL source code from the proprietary software component.

Once the developer reworks the code, the software component will be scanned again to verify that the GPL code has been removed before the ticket proceeds for legal review.

Scenario 3: Identified issue with linkages

In this scenario, the compliance ticket has passed legal review and is now in the architectural and linkages review. The compliance officer discovers a linkages issue. In this case, the compliance officer moves the compliance ticket back into the resolution phase, assigning it to a developer to resolve the linkage issue.

Scenario 4: Source code no longer used

In this scenario, Engineering decides that a software component is not going to be included in the product while the software component is in transit through the compliance process. As a result, its compliance ticket is closed (rejected). The next time this component is going to be used, it must go through the compliance process and progress as approved before it is integrated in the product or service source code base.

Scenario 5: IP at risk of requiring release

In this scenario, legal review uncovered that closely held intellectual property has been combined with open source code. The legal counsel will flag this and reassign the compliance ticket to engineering to remove the proprietary source code from the open source component. In the event that engineering insists on keeping the proprietary source code in the open source component, the OSEC will have to approve the release of the proprietary source code under an open source license.

Scenario 6: Unresolved issue found

In every case, when an OSRB member discovers a compliance issue in the software component, the component goes through the same life cycle:

- Engineering fixes the identified issue.
- Auditing team re-scans the source code and provides a new report.
- Legal examines the new audit report.
- Compliance office ensures that there are no open issues in the architecture and linkage analyses.

Scenario 7: Source code is approved

Once a software component has received the audit, legal, and compliance approvals, it will be reviewed during an OSRB meeting. If nothing has changed in its status — that is, it is still in use, is the same version, and has the same usage model (Figure 18) — the compliance officer will:

- Update the software inventory to reflect that the specific open source software version x is approved for use in product y, version z.
- Issue a ticket to the documentation team to update end user notices in the product documentation, to reflect that open source is being used in the product or service.
- Trigger the distribution process before the product ships.

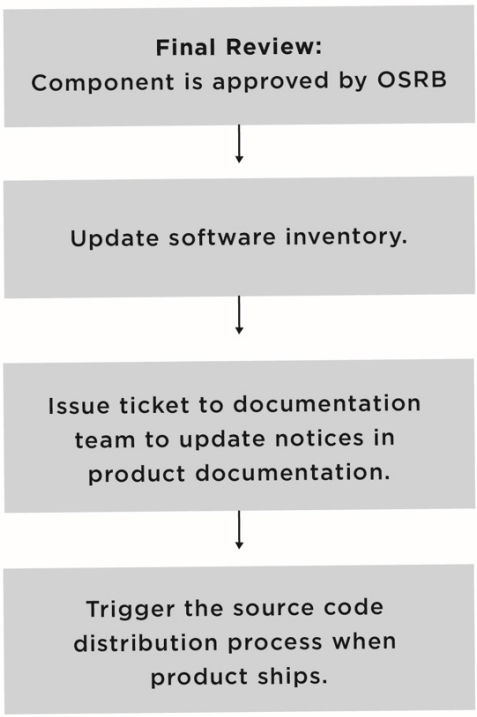


Figure 18. Steps accomplished after the OSRB approval

Scenario 8: Source code is rejected

In this scenario, the OSRB decides to reject usage of a specific software component. There are several reasons that might lead to such a rejection:

- The software component is no longer in use.
- There are linkage issues that cannot be resolved easily. The decision is to stop development and design a better solution.
- There are license incompatibility issues that cannot be resolved easily.
- There are intellectual property issues preventing the use or release of the specific component.
- Other reasons.

INCREMENTAL COMPLIANCE PROCESS

Incremental compliance is the process by which compliance is maintained when product features are added to a baseline version that has already completed initial compliance. (Initial compliance, also called baseline compliance, happens when development starts, and goes until the release of the first version of the product.) Incremental compliance requires a comparatively small effort in comparison to the efforts involved in establishing baseline compliance.

Figure 19 illustrates product development and incremental compliance.

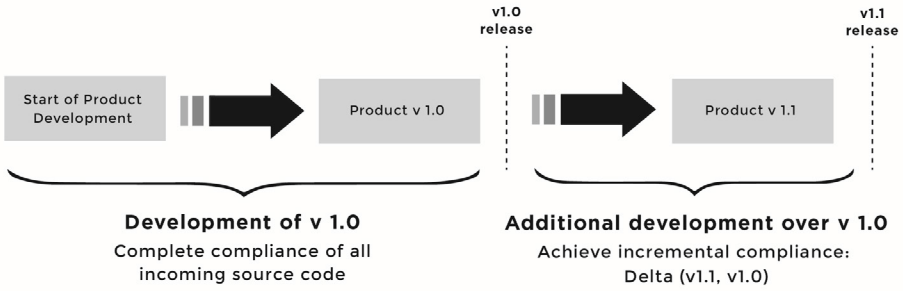


Figure 19. Incremental compliance

In this example, the compliance team identifies all open source included in the software baseline (here called v 1.0), and drives all of the open source components through the compliance process. Once the product ships, development begins on a new branch that includes additional features and/or bug fixes — in this example, v 1.1.

Several challenges arise with incremental compliance. Specifically, you must correctly identify the source code that changed between version 1.0 and version 1.1, and verify compliance on the delta between the releases:

- New software components or snippets may have been introduced.
- Existing software components or snippets may have been retired.

- Existing software components may have been upgraded to a newer version.
- The license on a software component may have changed between versions.
- Existing software components may have code changes involving bug fixes or changes to functionality and architecture.

The important question is “How can I keep track of all of these changes?”. The answer is simple: deploy a bill of material difference tool (BOM diff tool), as discussed in Chapter 7. Briefly, for the purpose of this discussion, the tool identifies the delta between two BOMs for the same product or software stack. Given the BOM for product v1.1 and the BOM for v1.0, the tool computes the delta, and the output of the tool is the following:

- New software components added in v1.1.
- Updated software components (differences were found in the code bases of the older and current release).
- Retired software components.

With this information, achieving incremental compliance becomes a relatively easy task:

- Added software components should enter the compliance process.
- Compute a line-by-line diff of the source code in changed software components, and decide if a new source code scan is required or not.
- Update the software registry by removing the software components that are not used anymore so they will not appear in the open source notices for the new software release.

Figure 20 illustrates the incremental compliance process. The BOM files from the two releases (to be compared) are pulled from the build server, each corresponding to a different release. The BoM tool computes the delta to produce a list of changes as previously discussed. At this point, the compliance officer will create new compliance tickets for all new software

components and snippets in the release, update compliance tickets where source code has changed and possibly re-run them through the process, and finally update the software registry to reflect retired software components from the new release.

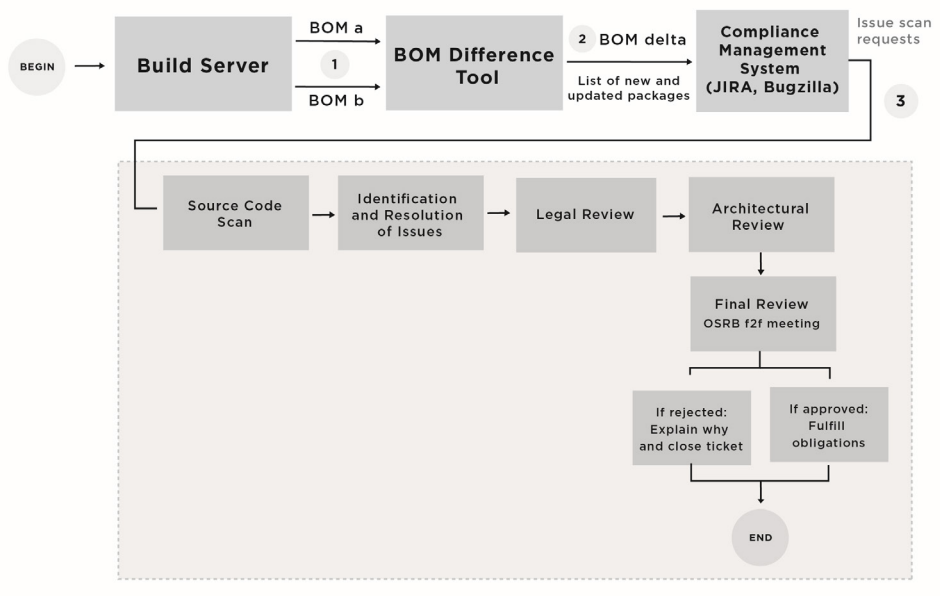


Figure 20. Example of an incremental compliance process

OSRB USAGE FORM

Completing the usage form is one of the most important steps when bringing open source software into a company (ingress), and should be taken very seriously. Developers fill out the online form requesting approval to use a given open source software. The form comprises several questions that will provide necessary information for the OSRB, allowing it to approve or disapprove the usage of the proposed open source component.

Table 6. Information requested as part of the OSRB usage form

Field	Description
Submitter Information	Company ID of employee submitting the form (facilitating retrieval of employee name, phone, email, manager, location, and team from a company directory)
Open Source Code Information	Package name, version, website, short description Software class: open source, internally developed, third- party License name, version, and license website Software category: OS/kernel, driver, middleware, library, utility, other (explain), etc. Benefits of using the OSS component Alternatives to using the component/package Consequences of not using the software Location of the software in the SCMS
Use Case	Internal (tools, IT, etc.) Distributed as part of a product Enabling an outward-facing service
Modification	Modified (Y/N)? Includes company IP? Exposes IP?

Table 6 highlights the information requested in a usage form. Usually, these values are chosen from a pull-down menu to make the data entry efficient.

Note on Downloaded Open Source Packages

It is vital to archive open source packages downloaded from the web in their original form. These packages will be used in a later stage (prior to distribution)

to verify and track any changes introduced to the source code by computing the difference between the original package and the modified package.

If a third party software provider uses open source, the product team integrating that code into the product must submit an OSRB usage covering the open source code used by the third party software provider.

If the third party software provider delivers only binaries, not source code, then the company must obtain a confirmation (for instance, a scan report) from them in a disclosure format that list all open source code used in those binaries along with a statement that insures compliance in terms of notices and source code availability (when applicable).

Note on Architecture Diagram

The architectural diagram illustrates the interactions between the various software components in an example platform. Figure 21 (next page) provides an architectural diagram template that companies can use to model the interactions of various software components and understand their relationship. It covers:

- Module dependencies
- Proprietary components
- Open source components (modified versus as-is)
- Dynamic versus static linking
- Kernel space versus user space
- Shared header files
- Communication protocols
- Other open source components that the software component in question interacts with or depends on, especially if it is governed by a different open source license

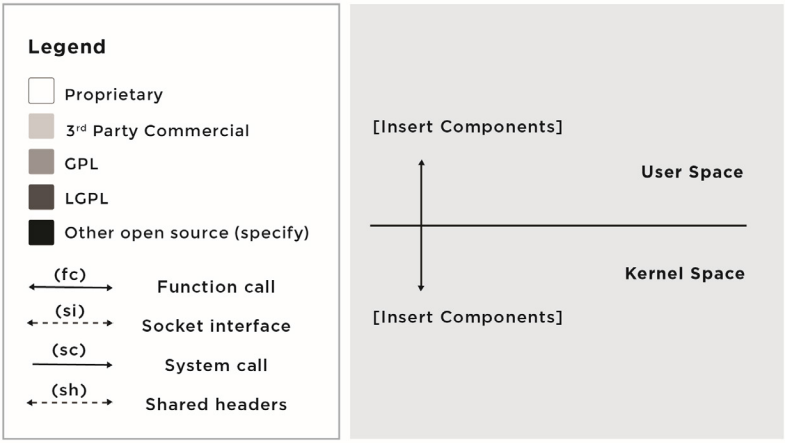


Figure 21. Template for architectural diagram (applies to an embedded environment that relies on C or C++)

Rules Governing the OSRB Usage Form

There are several rules governing the OSRB usage form. Here are a few:

- The form applies only to the usage of open source in a specific product and in a specific context. It is not a general approval of the open source code for all use cases in all products.
- Developers should update and resubmit the usage form whenever the plans for that open source component or snippet change.
- The OSRB must approve the form before engineering integrates the open source code into the product build.
- The form is the basis of audit activity and provides information the OSRB needs to verify if the implementation is consistent with the usage plan expressed in the form, and with the audit and architectural review results.

AUDITING

Good audit practices ensure a thorough understanding of the provenance of all software that will be incorporated in a product, software stack, or a web based service. With this understanding comes the ability for your organization to meet its open source software license obligations.

The auditing policy is usually simple and straightforward: All source code must be audited and have an audit report attached to its compliance ticket. The audit process consists of the workflow that requests follow after engineering has submitted an OSRB usage form for a specific software component or snippet.

Figure 22: Basic audit process

The audit process illustrated in Figure 22 (previous page) consists of the following phases:

- Receive the OSRB usage form, which includes the location of the source code to audit.
- Perform a scan of the source code.
- Perform analysis of components and snippets flagged by the scanning tool as unknown or unclear source and/or license. These typically need to be identified manually by a compliance engineer.
- Produce final report covering the software BoM for all scanned code.

SOURCE CODE DISTRIBUTION

The distribution policy and process have two primary goals:

- Inform customers that the product or the service contains open source software, and notify them of their rights to receive the source code when applicable.
- Ensure the correctness of the source code distributed in fulfillment of open source license obligations.

Distribution Incentives

There are three major business incentives to distribute open source code:

- Meeting license obligations,
- Contributing enhancements to an open source project, and
- Creating and contributing code to a new open source project.

In the next sections, we explore each of these cases and the motives driving the distribution or publication of the source code.

Meeting License Obligations

In this instance, the organization has incorporated open source software into a product or service, and due to the licenses involved, they now have the obligation to make source code available, including any modifications to it. The organization then compiles its open source BoM, and publishes all notices and required source code. We informally refer to this distribution model as a one-way distribution, versus a two-way that involves full community interaction to incorporate the modification back into the upstream branch.

Contributing modifications to an existing open source project

In some cases, the open source license does not include an obligation to make modifications available for license compliance purposes. However, organizations may consider releasing their modifications and possibly

upstreaming them to reduce technical debt, or, in other words, to reduce the cost of maintaining those modifications internally and to remain synchronized with the upstream branch. This is an example of a source code distribution (or publication) that is not motivated by compliance obligations.

Creating a new open source project

Organizations may have a business need to create a new open source project and contribute source code to it. This case is different from contributing source code (in the form of bug fixes or new feature implementation) to an existing open source project.

Distribution Policy and Process

The goal of a distribution policy is to govern the process of supplying source code and to provide guidelines on the various logistical aspects of meeting open source license obligations. The distribution policy applies to any software where the license requires redistributing source code, and it covers the publication process, publication methods, modes, and checklists.

Prior to triggering the distribution process, you need to decide on the method and mode of providing source. Subsequently, the process begins with preparing source code for external distribution, following a pre-distribution checklist, ensuring source code package availability, and then completing the post- distribution checklist.

Figure 23 illustrates a sample distribution process. It includes:

- Deciding on the source code distribution method
- Deciding on a distribution mode
- Preparing the source code packages for distribution
- Completing a pre-distribution checklist to ensure that all prior steps have been successfully completed and that the source code packages are ready for external distribution

- Executing the distribution by hosting the source code packages on a publically accessible web site
- Completing a post-distribution checklist to capture any possible errors that took place as part of the distribution process

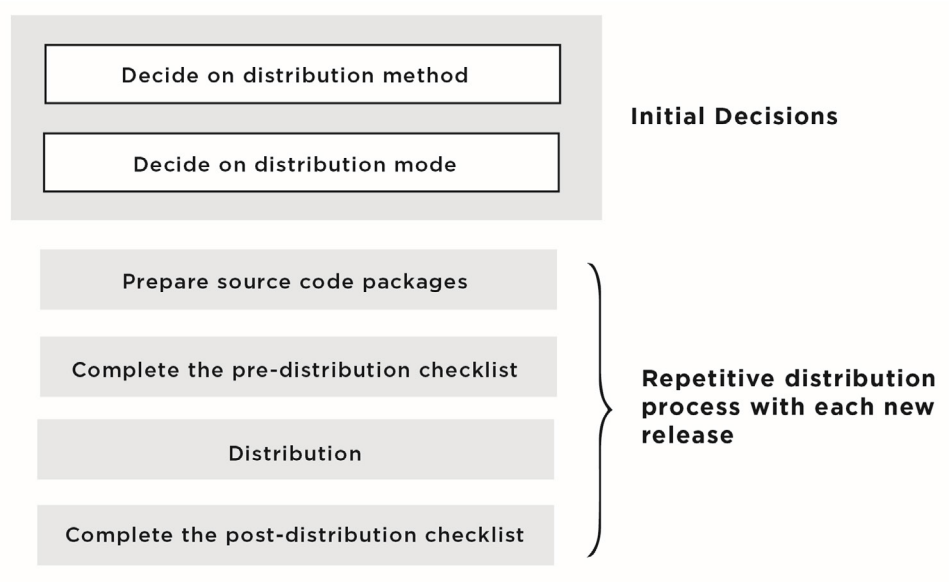


Figure 23. Sample distribution/publication process

Distribution Methods and Modes

There are three main distribution methods for making source code packages available: instant compliance, online distribution and on-demand distribution. We will discuss each of these methods in the following sections.

Instant Compliance Method

Following this distribution method, you provide the code when or shortly after your product or software stack ships, and make it made available to anyone who wishes to get it, typically via a website download. This is the preferred method of distribution for developers (and compliance enforcers), as it gives them direct access to the source code without having to pass the eligibility requirement (i.e., for instance they do not need to buy the product to be eligible to receive the source code).

There are two possible drawbacks to consider when choosing this method of distribution. Firstly, depending on the amount of open source code involved, it may require a lot of effort to package all source code and make it available on the website by the date the product ships, at a time when all hands are on deck getting the product out the door. Secondly, you will be building up an expectation that future code distributions will use this method as well. This is a very high expectation to meet every single time.

Online Supply Method

Following this distribution method, you provide exclusive access to your customers only, as they are the only entities eligible to receive the source code. Companies can manage this method via a secure website that requires a certain authentication on the part of the client to access and download the source code packages.

On-demand Distribution

The on-demand distribution method is a variation of the online supply method in which companies rely on the written offer (in the case of the GPL/LGPL family of licenses) to communicate to their clients how they can request or access the source code. Some organizations prefer to have

a written request, sent to a corporate email address or a postal address (specified in the written offer), the result of which is that the client receives a copy of the source code, after the verification of their eligibility.

This compliance method often gives the organization additional buffer time to finish packaging the source code after the product has shipped. However, it is not regarded as a preferred method of distribution, due to the overhead of verifying the eligibility of the people asking to access the source code and the resources required to fulfill the requests. In addition, with the specific cases of the GPL/LGPL family of licenses, the written offer to access the source code must be valid for three years. Therefore, you will need to administer the distribution of the code for at least three years from the date you have last shipped your product. If you opt for providing the source code on a CD-ROM/DVD, this will introduce some additional cost to you and additional verification steps to ensure the correctness of the process of burning the CD-ROM/DVD containing the source code packages.

Distribution Checklists

There are many checkpoints for validating the open source packages before they make their way to the website for public consumption. Moreover, validation is required after source code becomes publicly available. Below we outline the checklist of the pre- and post- distribution process.

Pre-Conditions for Distribution

The following is a list of conditions that should be met before source code packages are ready for distribution:

- The open source package or snippet has been approved for use.
- The product containing the open source software is ready to ship or has shipped already.
- If you are publishing GPL/LGPL code available, ensure you are providing and documenting the modifications you have introduced.

You have performed a linguistic review. Although this is not compliance-related, there have been issues in the past related to future product code-names used, obscene or vulgar language, and references to individuals, email addresses, and/or internal URLs left in the code. All such mentions should be cleaned up.

Pre-Distribution Checklist

The following is a sample checklist to follow before publishing or distributing source code:

- Verify that Engineering have documented their modifications to open source package as part of the change log.
- Ensure that each modified source code file contains an additional entry for a copyright notice, disclaimer, and a generic "change log" entry.
- Confirm that Engineering has reviewed the contents of the source code package have been reviewed by Engineering.
- Verify that the product manual includes all the open source notices.
- Ensure that the open source package compiles on a non-corporate machine to guarantee that the open source package you are about to distribute compiles on a vanilla, end-user system.
- Indicate how to access the code of the open source package (written offer) either through a web page download or by contacting your company via email or postal mail at a specified address provided in the product manual.
- Verify that the written offer is sufficient to cover all parts of the source code that require such an offer (principally, code licensed under any of the GPL/LGPL family of licenses).
- Perform a linguistic review to remove inappropriate comments from the source code. Some companies forget to go through a linguistic review and when/if their product is hacked, they face embarrassment from exposure of inappropriate comments left in the source code. Another important reason to perform a linguistic review

is to ensure that the source code and comments do not refer to future product code names or capabilities.

- Ensure that existing license, copyright, and attribution notices are not disturbed.
- Verify that the source code to be distributed corresponds to the binary that goes with the product, that the source code builds into the same library distributed with the product, and that build instructions are included in the source distribution (derived binaries are usually identical except for time/date stamp).
- Verify that the package adheres to the linkage relationships and interactions defined in the OSRB usage form. For instance, if the developer declared that they would dynamically link that component to an LGPL licensed library, then we need to verify they have done so, and have not used a static linkage method instead. The compliance team can do this verification by using a linkage dependency-mapping tool.
- Ensure a copy of the license text is already present in a LICENSE file in the root folder of the open source package.
- If the source code package requires special build tools or environment, then include the details in a README file or similar.

Post-Publication Checklist

The following is a sample checklist to go through after publishing source code:

- Source code packages have been successfully uploaded to the website and can be downloaded on an external computer.
- Source code packages can be uncompressed on an external computer without errors.
- Source code packages can be compiled and built on an external computer without errors.

Written Offer

Below is an example of a written offer to provide the source code:

> *To obtain a copy of the source code being made publicly available by FooBar, Inc. ("FooBar") related to software used in this FooBar product ("Product"), you should send your request in writing to:*
>
> *FooBar Inc.*
>
> *Attention: Open Source Compliance*
>
> *Street Address*
>
> *City, State, Postal Code Country*
>
> *FooBar makes every possible effort to make the source code publicly available at http://opensource.foobar.com ("Website") within reasonable business delays. Before sending your written request, please check the Website, as the source code may already be published there.*

Alternatively, if you prefer receiving requests via email and not via postal mail, the wording of the written offer would change slightly to the following:

> *To obtain a copy of the source code being made publicly available by FooBar, Inc. ("FooBar") related to software used in this FooBar product ("Product"), you should send your request in writing to opensourcecompliance@foobar.com.*
>
> *FooBar makes every possible effort to make the source code publicly available at http://opensource.foobar.com ("Website") within*

reasonable business delays. Before sending your written request, please check the Website, as the source code may already be published there.

Chapter 6

RECOMMENDED PRACTICES

This chapter highlights some of the recommended practices and various considerations when integrating open source in commercial products and services. The chapter consists of three parts:

- Recommended practices that map to the various steps within an open source compliance process.
- Compliance considerations in relation to source code modifications, notices, distribution, software design, usage, linkages, and code mixing.
- Recommended practices related to the various building blocks in an open source compliance program.

COMPLIANCE PROCESS

As a refresher, the compliance process includes the various steps a software component goes through before the OSRB approves it for inclusion in a product or a software stack. The process starts by identifying the various software components integrated into the product's build system, and ends by compiling a list of resulting license obligations.

The following sections provide recommended practices for processing compliance requests. The recommended practices map directly to the steps illustrated in the compliance process (Figure 24, next page).

Identification Phase

In the identification phase of a compliance process, organizations identify all of the components or elements entering the build system, origin, and license information. There are three main sources for incoming source code:

- Proprietary software created by internal developers, which may include snippets of open source or which integrates open source at the component level, with dependencies on or links to open source code
- Third party software developed by independent providers or consultants and made available under a commercial or open source license. This software category may include snippets or dependencies as above.

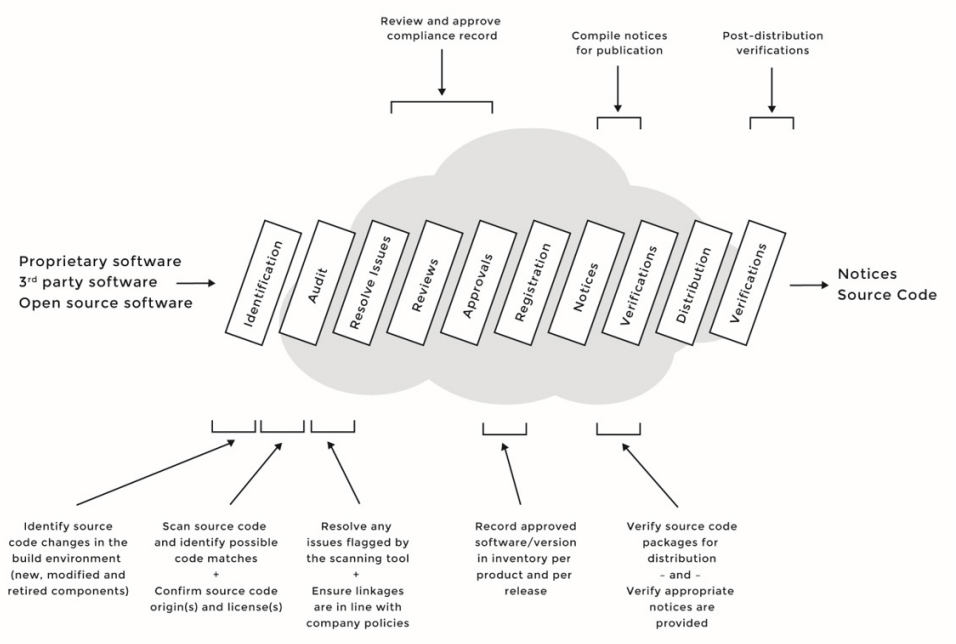


Figure 24. Compliance end-to-end management process

We recommend that you identify all incoming software components and run them through the compliance process.

Source Code Auditing

There are three core recommended practices for source code auditing or scanning:

Scan all source code

Scan all source code incorporated into products and services, because development teams may have introduced open source into proprietary or third party source code. Furthermore, development teams may have made modifications to open source components, triggering the need for additional due diligence and potential additional obligations. Therefore, it is critical to audit and identify all source code included in the software stack of a product or a service.

Scan newer versions of previously approved packages

Since compliance is verified on a product-by-product, service-by-service basis, approving for use in one case does not necessarily serve for all cases. As a rule, each time developers modify a previously approved component or plan to use a previously approved component in a different context, the compliance team should rescan the source code, and the component should pass the approval process again.

Furthermore, license changes can occur between version upgrades. When developers upgrade versions of open source packages, make sure that licenses of new versions are the same as those employed with older versions.

Scan early and often

The open source development model encourages frequent releases, starting on project day one, to give users opportunities to experiment and report bugs. The goal is to make quality assurance activities a regular part of the development process. “Scan early and often” follows the same spirit. Scanning source code early in the development process and continuing to do so regularly ensures that compliance efforts are not lagging behind the development efforts. Organizations should also create a list of conditions that define when a new scan is required, to make the process more efficient.

The “scan early and often” approach has several advantages:

- It aids in the discovery of compliance issue very early in the process.
- It accelerates providing solutions to discovered problems within

acceptable timeframes without posing a serious threat to the delivery timeline.

- It improves the efficiency of processing incremental, scans since it reduces the delta of source code that needs to be scanned from previous source code scans.

Resolving Issues

When source code is scanned and compliance issues are discovered and flagged, there are a number of ways to resolve issues:

- When in doubt with the scan results, discuss with engineering staff.
- Inspect and resolve each file or snippet flagged by the scanning tool.
- Identify modifications introduced to open source software. Ideally, you should not rely on engineers to remember if they made code changes (let alone document them). You should rely on your build tools to identify code changes, who made them, and when.
- If, for instance, the scanning tool identifies the use of un-approved GPL licensed source code (a snippet) within a proprietary component, this will be reported to Engineering with a request for correction. It is highly recommended to re-scan the source code after Engineering has resolved the issue, to confirm removal of the problem source code and its replacement with appropriate and compatible code.
- In preparation for the legal review, it is best to provide attorneys with all discovered licensing information for specific components including the source code audit report generated by the scanning tool, the COPYING, README, or LICENSE files for open source components, and the licensing agreement for software components received from a third party software provider

Architectural Review

The architecture review is an analysis of the interaction among open source, proprietary, and third party software components. Companies often

conduct architecture reviews with the architect responsible for the product in question, plus developers responsible for the various key software components.

The goal of this review is to identify:

- Components that are open source (used as-is or modified)
- Proprietary components
- Third party components under a commercial license
- Component dependencies
- Communication protocols among components and subsystems
- Dynamic versus static linking (discussed in the following section)
- Components deployed in kernel space (drivers, etc.) versus user space (libraries, middleware, applications)
- Components that use shared header files
- Other open source that the specific software component interacts with or depends on, especially if it is governed by a different open source license

The result of the architecture review is an analysis of the licensing obligations that may extend from open source to proprietary or third party components.

Approvals

As part of the approval step in the compliance process, there are two main recommended practices:

- Verify that all subtasks related to the compliance ticket have been completed and closed before approving the compliance ticket. It's easy to forget subtasks or pending sub-issues, but doing so may lead to prematurely closing a compliance ticket even though open issues remain.

- Record a summary of discussions that lead to approval or denial. Such documentation can prove very useful when attempting to determine on which basis the OSRB has approved the use of specific component or snippet and how issues were resolved.

Notices

Organizations using open source in products and services need to:

- Acknowledge the use of open source by providing full copyright and attribution notices.
- Inform end users how to obtain a copy of the open source code (when applicable, for example in the case of GPL and LGPL licensed source code).
- Reproduce the entire text of the license agreements for the open source code included in the product.

Some recommended practices in this area include:

- Collect attribution and license notices incrementally, as open source is approved for inclusion. Following this method, the required notices file will always be up to date and will include lists of all open source, license information, copyright, and attributions notices.
- Use clear language in the written offer and be inclusive of all open source included in the product.
- Ensure that the end users of the product know how to locate this information, whether on the product itself, in the product documentation (user manual or CD-ROM/DVD), and/or on a website.

Verifications

It is very helpful and efficient to develop, maintain, and evolve checklists that cover the verification steps that the compliance team follows, both to ensure consistency and to ensure that no verification steps are overlooked. Examples of pre-distribution verifications include:

- Open source packages destined for distribution have been identified and approved
- Inappropriate comments have been removed from the source code packages (this is not strictly a compliance issue; however, comments may reveal a compliance issue that is not as visible)
- Source code packages made available (including modifications) match the binary(ies) shipping in the product or software stack
- Appropriate notices have been included in the product documentation, in addition to the availability of a written offer to inform end users of their right to request source code for identified open source (when applicable)

Once open source packages are uploaded to the distribution website (and/or stored on equivalent media), your work is not complete. You still need to verify that:

- Packages have been uploaded correctly
- Packages can be downloaded and uncompressed on an external computer without error
- Included packages compile/build properly
- Developers did not leave comments about future products, product code names, mention of competitors, or any inappropriate comments

TOOLS AND AUTOMATION

Tools are an essential element in a compliance program in that they can help organizations perform compliance activities efficiently and accurately. Many tools can prove very useful in an open source compliance program:

- Source code scanning and license identification tools
- Project management tools
- Bill of material difference tools
- Linkage analysis tools

In the following sections, we provide basic information about such tools and how they fit within the compliance context. Multiple commercial/proprietary and open source tools provide the various functionalities described below.

Source Code Identification Tools

Source code scanning and license identification tools provide recognition and analysis capabilities to assist users in identifying the origin of source code and licenses associated with open source software components and snippets.

There are several providers for such tools that we list here in alphabetical order:

- Antepedia (**http://www.antepedia.com/**)
- Black Duck Software (Synopsys) (**https://www.blackducksoftware.com/**)
- Flexera Software (**https://www.flexerasoftware.com/**)
- FOSSA (**http://fossa.io/**)
- FOSSID AB (**http://www.fossid.com**)
- nexB (**https://www.nexb.com/**)
- Protecode (Synopsys) (**http://www.protecode.com/**)
- Rogue Wave Software (**https://www.roguewave.com/**)

- WhiteSource Software (**https://www.whitesourcesoftware.com/**)

In addition to commercial tools, there are two prominent open source compliance tools:

- Binary Analysis Tool (**http://www.binaryanalysis.org/**)
- FOSSology (**https://www.fossology.org/**)

Project Management Tools

A project management tool is essential to managing and tracking compliance activities. Some companies use bug tracking tools (see list below) already in place with a customized compliance workflow; other companies rely on specific project management tools or even in-house solutions. Whatever your preference, tools should reflect the workflow of compliance processes, facilitating moving compliance tickets from one phase of the process to another, providing task and resource management, time tracking, email notifications, project statistics, and reporting capabilities.

Example bug-tracking tools commonly employed for customizing a compliance workflow and tracking compliance tickets include Bugzilla (**https://www.bugzilla.org/**) and JIRA (**https://www.atlassian.com/software/jira**).

Software Bill of Material (BOM) Difference Tools

The goal of a Software BOM difference tool is to compute the difference between two BOMs and produce a list of changes. Such a tool enables efficient incremental compliance when facing newer versions of an existing base code (for instance, going from release 1.1 to 1.2).

The inputs to a BOM difference tool are two BOM files that represent the list of components available on two different versions of a product or service code base.

The output of the BOM difference checker documents the list of new components, retired components, and modified components.

BOM management tools are plentiful in the world of physical manufacturing, but less so for managing use of open source software. In this author's experience, BOM difference tools that support open source management processes are usually homegrown and/or built as mash-ups of existing tools and capabilities. Depending on the form and format of the bill of materials, it is possible to use command-line diff tools, productivity tools (spreadsheets, etc.), directory comparison tools, and reports from build and continuous integration tools, plus scripting "glue," to create web-based BOM version comparisons. Figure 25, created for illustration purposes, shows the sample output of a homegrown BOM difference tool.

BOM Difference Report

Package	Image	Owner	Version	Submission
adapterbase	none	chad.everett@acme.co.nz > richardburton@acme.co.nz	1.0.0	23
adec-omxump3-msm7x25	customization	pamela.anderson@acme.co.nz	1.0.0	1 > 2
amazonservice	luna	una.due@acme.co.nz	1.0.0	4 > 5
audiod	luna	damien.lewis@acme.co.nz	1.0	111 > 114
audiod-config	luna	damien.lewis@acme.co.nz	1.0	70 > 72
browser-adapter	cust	chris.reeve@acme.co.nz	1.0.01	118 > 131
browser-service	cust	chris.reeve@acme.co.nz	1.0.01	126 > 128
bzip2	rockhopper	> eric.idle@acme.co.nz	1.0.2	1
camd-omap	any	itzaak.periman@acme.co.nz	1.0.0	S > 7
cifs	rockhopper	> eric.idle@acme.co.nz	3.0.23c	1
clam	any	sean.connery@acme.co.nz	1.0.0	7 > 8

Figure 25. Example BOM difference report

Linkage Analysis Tool

The goal of the linkage analysis tool (also called dependency checker tool) is to flag problematic code combinations at the dynamic and static link level, specific to C and C++ programming languages. The tool identifies a linkage conflict between the license of the binaries and the license of the libraries it links to, based on predefined license policies that the user of the tool has already defined.

There are many tools that can be used together to fulfill the function of dependency checking.

The main requirements for dependency mapping are the abilities to:

- Identify linking among binaries and libraries
- Identify licenses for binaries and libraries
- Connect with license scanning tools or consume the output thereof
- Configure to match company policy preferences to flag linkages that violate those policies (e.g., linking with GPL-licensed code)

In this our experience, linkage analysis tools, much like BOM tools, are usually homegrown and/or built as mash-ups of existing tools and capabilities. One off-the-shelf open source tool of this type is The Linux Foundation Dep-Checker (**http://git.linuxfoundation.org/dep-checker.git/**).

CHAPTER 7

MANAGING COMPLIANCE INQUIRIES

This chapter presents guidelines for handling compliance inquires. These guidelines aim to maintain a positive and collaborative attitude with requesters while investigating allegations and ensuring proper actions when violations actually occur.

Several organizations have received negative publicity and/or have been subject to legal action after ignoring requests to provide compliance information; did not know how to handle compliance inquires; lacked or had a poor compliance program; or simply refused to cooperate, thinking (incorrectly) that license terms were not enforceable. Today, best practices inform us that none of these approaches is beneficial to any party involved. Therefore, companies should not ignore compliance inquiries — rather, they should acknowledge receipt of inquiries, inform inquiring parties of pending response, and provide an estimated date for follow-up.

Compliance inquiries can include requests for:

- Access to source code in accordance with a written offer to provide source code licensed under GPL, LGPL, or other licenses
- Access to source code for an undisclosed component that was discovered in a product
- Verification of whether a specific open source component is used in a product or service
- Update to an out-of-date attribution or copyright notice
- Providing files missing from open source packages made available as part of license obligations

Companies usually receive compliance inquiries through a dedicated email address that they advertise in their written offer or as part of their open source notices.

RESPONDING TO COMPLIANCE INQUIRIES

This section introduces a method for responding to compliance inquiries. Figure 26 presents a simple process used to respond to compliance inquiries. The process consists of six steps that we discuss in the following sections.

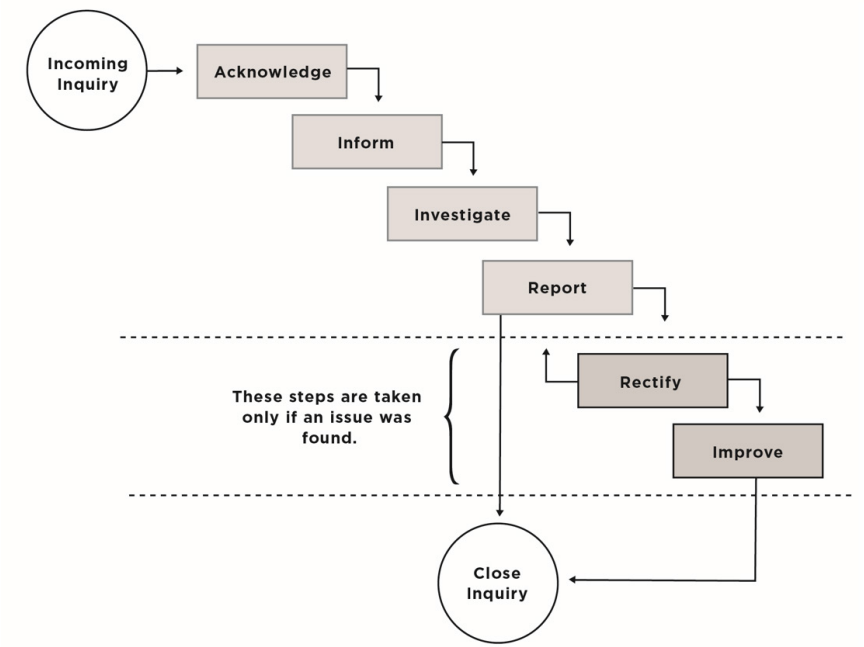


Figure 26. Process of responding to compliance inquiries

Acknowledge

Once you receive the compliance inquiry, you should respond immediately, confirming receipt and committing to investigate by a specific date. It is important to understand the inquirer's identity and motive and to verify whether the complaint is justifiable, accurate, and current. It is important to realize that some inquirers often do not fully understand licenses, leading to mistaken assumptions and submissions. If an inquiry is missing information, request additional clarification, such as:

- The name of the affected product(s) or service(s) or the exact code of concern
- The reason why a violation is believed to exist
- The name of the project code and license that may have been violated
- A link to the project site

Inform

We recommend maintaining an open dialog with inquirers. Always highlight your open source compliance practices and demonstrate historical good faith efforts toward compliance. Inform inquirers about your compliance program and practices, and assure them that you will investigate their question. It is also advisable to send updates of your internal investigation as they become available.

Investigate

In this step, investigating reported allegations, you should refer to the compliance record for the component in question, review it, and verify if and how the compliance record compares with the inquiry.

Report

After concluding the internal investigation within an acceptable delay, and creating an internal record of the findings, you need to inform the inquirer of the results.

Close Inquiry

If the compliance inquiry was a false alarm, you can close the compliance inquiry ticket without any further action (other than informing the inquirer of that resolution).

Rectify

If the investigation uncovers an actual compliance issue, you should report back to the inquirer confirming that fact, with assurances that your organization will take all the necessary steps to bring your product or service back to compliance, specifying a date by which you expect to complete this task. It is your responsibility to resolve the issue with the inquirer, while being collaborative and showing good will. You need to show that you understand the obligations under the applicable license, communicate how — and when — you will meet the obligations. Once you fix the problem, you should notify the inquirer immediately, and invite them to verify the solution.

Improve

If there was a compliance issue, you should call for an OSRB meeting to discuss the case, learn how this non-compliance occurred, and improve existing process and practices to ensure that such errors do not happen again.

General Considerations

There are two specific considerations to be mindful of. The first is to treat all inquiries as formal inquiries, and work under the assumption that any information you disclose as part of the interactions with the inquirer can become public. The second is to consider how your existing open source compliance efforts would measure up in an enforcement action, and work to improve your open source compliance program accordingly.

Enforcement Activities With Varying Motivations

There has been a recent increase in enforcement that is generally viewed as being not in the spirit of ensuring compliance, but with the goal of financial profiteering. Responding to financially motivated enforcement activities may be very different from compliance-motivated enforcement. Therefore, such activities may require a different approach and legal strategy than the one proposed in this chapter.

It is critical to correctly discern the difference between the two, as responding in the wrong way can make the process considerably more difficult and expensive. In the traditional compliance-motivated enforcement, the person bringing the complaint generally wants to see full and correct compliance with the open source license. In most cases, they are looking for assurances that the company will meet the terms of the licenses and make an internal process changes that ensure such future violations will not recur. They are generally well informed of their rights on their code and are willing to work with you to fix the issue so long as you are responsive and provide signs of progress. One of the worst mistakes a violator can make in this scenario is to take a hard line legal response, as this very quickly turns them from advocates to adversaries.

On the other hand, financially motivated enforcement is where the person bringing the complaint is looking for a payoff, generally through a quick financial settlement or (more rarely) punitive damages awarded by the court system. Unfortunately, these are not always initially distinguishable from compliance-motivated enforcement. However, a typical hint is that these types of enforcement actions are accompanied by an immediate fiscal settlement offer.

Your response to financially motivated enforcement depends upon many circumstances, not the least which being your own assessment of the complaint's validity, the deadlines in the letter/offer, and your legal counsel's advice. However, it is also important to consider the possibility that you are one of many companies being targeted in an organized manner.

Unfortunately, there are individuals who view this as a way to make easy money, and law firms who specialize in enabling them. If you suspect this is the case, we recommend a counsel-to-counsel discussion with other companies who may have faced similar situations.

Becoming a Gold or Platinum member of the Linux Foundation enables your in-house counsel to participate in Linux Foundation Legal Counsel events and conferences, and provides an opportunity for sharing of knowledge and best practices especially those related to managing financially motivated enforcement.

CHAPTER 8

OTHER COMPLIANCE-RELATED PRACTICES

This chapter highlights compliance best practices and various considerations outside of the actual compliance process.

EMPLOYEE APPRAISAL

There are four challenges that all companies face with regard to engineering and compliance enforcement:

- Ensuring engineers consistently fill out request forms for each open source component they want to use
- Requiring engineers to respond in a timely fashion to compliance tickets
- Verifying engineers are following the guidelines set by the OSRB
- Mandating engineers to take your internal open source compliance training

A practice that has proved to be effective in helping companies face these four challenges is to include open source and compliance metrics as part of employee performance reviews. As a result, part of the developers' yearly bonus will depend on the extent to which they have followed the compliance policies and procedures.

Reviewers may evaluate performance on whether employees have filled out OSRB forms for open source software they plan to use, responded to compliance tickets without significant delays, completed the open source and compliance training within the set time limit, and used open source within the guidelines set by the company's policy.

In turn, to use compliance as a factor in employee performance reviews, the OSRB must track these issues for each developer:

- Components that were included in the software BOM that don't have a corresponding approval

- Response time to compliance tickets
- Course completion
- Compliance violations reported to the executive team

WEB PORTALS

Some companies maintain both an internal and an external open source web portal. The internal portal hosts compliance policies, guidelines, training material, announcements, and access to related mailing lists. The external portal offers a consistent means of posting source code of open source packages they use, in fulfillment of license obligations.

MESSAGING

The single most important recommendation with respect to messaging is to be clear and consistent, whether internally — explaining company goals and concerns around open source, or externally — facing community participants. Having a community-facing site is particularly important when responding to compliance inquiries.

TRAINING

The goal of open source and compliance training is to raise awareness of open source policies and strategies and to build a common understanding of the issues and facts of open source licensing. Training may also cover the business and legal risks of incorporating open source in products. It also serves as a way to publicize and promote an organization's compliance policies and processes, and to promote a culture of compliance.

There are formal and informal training methods. Formal methods include instructor-led training courses where employees have to pass a knowledge exam to pass the course.

Informal methods include webinars, brown bag seminars, and presentations to new hires as part of the new employee orientation session.

Informal Training

Brown Bag Seminars

Brown bag seminars are usually presentations offered during lunchtime by either a company employee (in-house legal counsel, open source expert, compliance officer, etc.) or an invited speaker (most commonly a high profile open source developer). The goal of these seminars is to present and elicit discussions about the various aspects of incorporating open source in products or software stacks. These sessions can also include discussions of the company's compliance program, policies, and processes.

New Employee Orientation

In some instances, the Compliance Officer presents on organization compliance efforts, rules, policies, and processes to all new employees as part of the new employee orientation session. On their first day, new employees would receive a 30-minute training on open source and compliance. As a result, the new employees will have all the necessary

information they need, such as who are the internal subject matter experts, what intranet resources exist, and how to sign up for open source and compliance training.

Formal Training

Depending on the size of the organization and the extent to which open source is used in commercial offerings, the organization can mandate that employees working with open source take formal instructor-led courses and be tested on their subject-matter proficiency.

SOURCE CODE MODIFICATION CONSIDERATIONS

With respect to source code modification, we recommend to publish an internal set of guidelines in plain, non-legalistic language that establishes basic rules for modifying existing source code. For example:

- Source code modifications that will remain proprietary must not be made within an open source package, especially one that has derivative work obligations (e.g., GPL or LGPL).
- Proprietary source code must not link to an open source library that has a derivative work obligation. Companies usually request formal OSRB approval for such action.
- Ensure that any modifications to source code are documented in compliance with the open source license prior to distribution.
- All modifications to open source code modules shall be captured in the revision history of the module (change log file).

NOTICES CONSIDERATIONS

One of the key obligations when using open source is to ensure clear and accurate documentation of copyright, attribution, and license information, and the availability of a written offer (for GPL/LGPL licensed source code). The sum of all of these documentation obligations is often referred to as open source notices.

Companies using open source in their offerings must acknowledge the use of open source by providing full copyright attribution, and, in most cases, reproducing the entire text of the licenses of the open source software included in the product or service. Therefore, companies must fulfill documentation obligations by including copyright, attribution, and license notices text in the documentation of every product they ship and service they provide.

There are two primary options for fulfilling documentation obligations requirements:

- Display the open source notices on the product itself. This is a viable option if the product has a user interface that allows the user to interact with it and pull up or display licensing information. An example of this option is a cell phone or a tablet.
- Include the open source notices in the product manual or any kind of documentation accompanying the product.

Some companies opt for both options when possible, in addition to maintaining these notices on a given website (optional, but also often adopted, and it's low maintenance — basically just hosting the notices file on the website). The important takeaway from the notices considerations is to ensure that all open source notice requirements are satisfied prior to a product distribution or service launch.

DISTRIBUTION CONSIDERATIONS

Companies want to ensure that any source code subject to open source distribution obligations is compliance-ready prior to product shipment. By thoroughly integrating compliance practices into the development cycle, distribution considerations can be greatly simplified and streamlined.

USAGE CONSIDERATIONS

The following sections address considerations and caveats for using open source in a fully compliant manner.

Clean Bill of Materials (BOM)

Ensure that any inbound software does not contain undeclared open source. Always audit source code upon receipt from providers; alternatively, make it a company policy that software providers must deliver audit reports for source code they supply.

OSRB Form for Each Open Source Component

Fill out an OSRB usage request form for each open source component in use. Avoid using any open source without explicit OSRB approval.

Understand the Risks During Mergers and Acquisitions (M&A)

Understand the open source code in use and its implications as part of the due diligence performed prior to any corporate transaction. We discuss this topic in Chapter 13.

Retired Open Source Packages

If an approved open source package is no longer in use, engineers must inform the OSRB to update the open source inventory; alternatively, the OSRB will discover that the package is not used anymore when they run the BOM diff tool.

Major Source Code Changes

If an approved package went through a major change, inform the OSRB to re-scan the source code; alternatively, the OSRB will discover that the package has been modified when they run the BOM diff tool. A major change in the design or implementation often affects architecture, APIs, and use cases, and in some cases may have an impact on the compliance aspect.

Reference Original Source Code

Document the URL from which you downloaded the open source package in addition to saving an original copy of the downloaded package.

Upgrading to Newer Versions of open source

Ensure that each new version of the same open source component is reviewed and approved. When you upgrade the version of an open source package, make sure that the license of the new version is unchanged from the prior version, as license changes can occur between version upgrades. If the license changed, contact the OSRB to ensure that compliance records are updated and that the new license does not create a conflict.

Compliance Verification Golden Rule

Compliance is verified on a product-by-product, service-by-service basis: Just because an open source component or snippet is approved is one context, it does not necessarily mean that it is approved for use in a different context, another product or service. New approval will be required.

Copy/Paste

Avoid using source code snippets, and avoid copying/pasting open source code into proprietary or third party source code (or vice versa) without prior documented OSRB approval. Such actions have serious implication on compliance.

Mixing Source Code with Different Licenses

Avoid mixing different open source licenses in a derivative work, as many open source licenses are incompatible with one another. It is highly recommended to seek legal support from your Counsel on this topic.

Source Code Comments

Do not leave inappropriate comments in the source code (private comments, product code names, mention of competitors, etc.).

Existing Licensing Information

Do not remove or in any way disturb existing copyrights or other licensing information from any open source components that you use. All copyright and licensing information must remain intact in all open source components, unless you are completely certain the license allows it to be changed.

ATTRIBUTION CONSIDERATIONS

Companies that include open source in a product need to provide required attribution to the end user. This section provides guidelines of how to fulfill open source attribution obligations.

Attribution Types

Open source attribution requirements differ from license to license, but we can generally group them into four categories:

Full License Text

A verbatim copy of the full license text is required for almost all open source licenses.

Copyright Notices

A verbatim copy of the copyright notices is required for many open source licenses.

Acknowledgments Notices

Some open source licenses explicitly require author attribution. In most cases, open source projects maintain a file called AUTHORS that includes the list of contributors; you can use this information as part of the attribution notice.

Information on Obtaining the Source Code

Most licenses with a source code redistribution obligation require either that the source code accompany the product or that the user receive a written offer with details on how to obtain the source code. The GPL and LGPL are examples of licenses in this category.

Presentation of Attributions

For each product or service containing or using open source, the attributions must be included in published user documentation (such as the product manual) distributed in printed or electronic form, such as a CDROM/DVD or a download from a website.

If products or services possess a graphical user interface or a command line administrative interface, you can also provide the option to display the attributions via that user interface.

For product updates such as over-the-air (OTA) updates for mobile phones, the attributions must also be revised when the product update includes new or updated open source components.

SPECIFIC LICENSE OBLIGATIONS

"Must include a copy of the license in documentation available to the end user"

The license of the open source component in question must be included in the user documentation for all products using this open source.

RECOMMENDATIONS

- In some instances, such as with mobile phones or tablets, manufacturers are able to provide the notices on the actual device via a web browser or a PDF viewer (i.e., licensing text is available on the device either in HTML or PDF format).
- For products with a user accessible file system, it is recommended that the license is included in the file system with a filename LICENSE to make it stand out and to be similar to the open source license filename.
- For product updates, license information must also be updated. For instance, when a new software release becomes available, the updated release must include an update license information file to reflect any open source changes introduced in the new release. Changes may include:
    - New open source used
    - Deprecated/removed open source
    - Open source upgraded to a new version, which may require updating the attribution/copyright notices, and, in some rare cases, updating the license

"Must include copyright notices in documentation available to the end user"

The license of the open source component in question may require including copyright notices in the product document available to the end user.

RECOMMENDATIONS

- For all products, copyright information must be included in printed documentation (such as a user manual).
- If the use case includes a graphical user interface, the end user should be able to view the copyright information from an ABOUT or a LICENSE screen.
- If the product has a user-accessible file system, the copyright information should be included in the file system in a file containing, for instance, all the copyright notices for all open source used in the product.
- For products updates, the copyright information must also be updated.

"Advertising materials may need special acknowledgments"

This advertising clause from the original BSD license is written as follows:

All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors.

Where applicable, all marketing and advertising material (including web-based, magazines, newspapers, flyers, etc.) must display the acknowledgement.

GENERAL GUIDELINES

You are probably already familiar with some of the guidelines that apply to open source licenses, such as not using the name of the open source project for endorsement, marking the source code modifications you have introduced, and preserving the original licensing, copyright, and attribution information. The following sections expand on these general guidelines in more detail.

No Endorsing or Promoting

You cannot use the name of the open source project, authors, or contributors in any marketing, advertising, or documentation (hard copy, digital, or on the web) without prior written permission.

Source Code Modifications Markup

When redistributing modified open source code, your modifications need to be clearly marked as such, including a copyright line for those modifications (company, year) while preserving the existing copyright lines.

Some companies elect a different approach — providing the original open source code along with the company's contributed patch files that apply against the original open source code. Following this approach, the company's modifications are clearly separated from the original open source code.

Preserving Original License, Copyright, and Attribution

Whenever you are redistributing open source code, with or without modifications, you must preserve the original licensing information, copyright lines, and other attributions.

Source Code Comments

Do not leave any inappropriate comments in the source code, such as private comments, product code names, mention of competitors, etc.

Existing Licensing Information

Do not remove or in any way disturb existing open source licensing copyrights or other licensing information from any open source components that you use. All copyright and licensing information is to remain intact in all open source components.

Chapter 9

SCALING OPEN SOURCE LEGAL SUPPORT

Open source compliance is often more of an operational and logistical challenge than a legal challenge. Achieving compliance requires the proper policies and processes, training, tools, and proper staffing that enable an organization to effectively use open source and contribute to open source projects and communities, all while respecting copyrights of their respective holders, complying with license obligations, and protecting the organization's intellectual property and that of its customers and suppliers.

However, legal counsel plays an indispensable role in supporting the open source compliance programs and core teams that most organizations create to ensure proper compliance.

In this chapter, we look closely at the role of the Legal Counsel in ensuring open source compliance, and offer practical advice that a Legal Counsel can provide to the development team. Such practical advice will enable software developers to make day-to-day decisions related to open source licenses without having to go back to Legal Counsel for every single question.

PRACTICAL LEGAL ADVICE

Practical advice from Legal Counsel to software developers may include:

- License Playbooks: Easy-to-read, digest-form summaries of open source licenses intended for software developers
- License compatibility matrix: A grid to help determine whether License-A is compatible with License-B. Software developers can use such a matrix as they merge incoming code from different projects under different licenses into a single body of code.
- License classification: An easy way to understand the different licenses, and the course of action needed when using source code provided under these licenses

- Software interaction methods: A guide to understanding how software components available under different licenses interact, and if the method of interaction is allowed per company compliance policies
- Checklists: A consistent, foolproof way to remember what needs to be done at every point in the development and compliance processes

In the following sections, we examine these five pieces of advice, provide examples, and discuss how they help software developers working with open source.

LICENSE PLAYBOOKS

License playbooks are summaries of commonly used open source licenses. They provide easy-to-understand information about these licenses, such as license grants, restrictions, obligations, patent impact, and more. License playbooks minimize the number of basic questions sent to Legal Counsel and provide developers with immediate legal information about these licenses.

Figure 27 (next page) provides an example license playbook for the GPL v2. Please note that we provide this playbook for illustration purposes only and its content should not be considered definitive.

SAMPLE PLAYBOOK FOR THE GPL v2.0

License Grant:

1. Copy and distribute verbatim copies of source code

2. Modify source code and copy and distribute copies of modified source code

3. Copy and distribute copies of object code

License Limitations:

Modified source code

Mark that source code has been changed (change log)

All source code

Mark with copyright notice

Mark with disclaimer of warranty

Keep intact with all other notices

Object code

Accompany with source code

Written offer to provide source code upon request

License Obligations:

Must include a copy of the license in documentation available to the end-user

Must inform user of how to obtain the source code and that it is covered by GPL v2

Must redistribute source code [including modifications, if any]

When redistributing source code, must include a copy of the license

Source code modifications must be clearly marked as such and carry a

Figure 27. Example license playbook for GPL v2 (for illustration purposes only)

LICENSE COMPATIBILITY MATRIX

License compatibility is the determination of whether a software component and its license are compatible with one or more other components and their licenses (i.e., that their licensing terms do not conflict). Compatibility also

addresses the appropriate licenses for works that combine two or more licenses (combined out-licensing).

License compatibility challenges can arise when combining diverse open source software components, in source and/or object form that are distributed under licenses with incompatible terms. The result of such combination is a licensing chimera, an aggregation of software components that for purely legalistic reasons cannot be redistributed.

An example of licensing incompatibility can be found in attempting to combine code distributed under the Apache version 2 license with software under the GNU GPL version 2.0 (due to patent termination and indemnification provisions not present in the older GPL license). An example of license compatibility is combining code licensed under the X11 license, which is explicitly compatible with the GPL version 2.

Figure 28 illustrates the creation of a single source component that originated from multiple sources under different licenses. In this scenario, you must ensure the sources have compatible license terms that allow you to join them in a binary or an object file without any conflict.

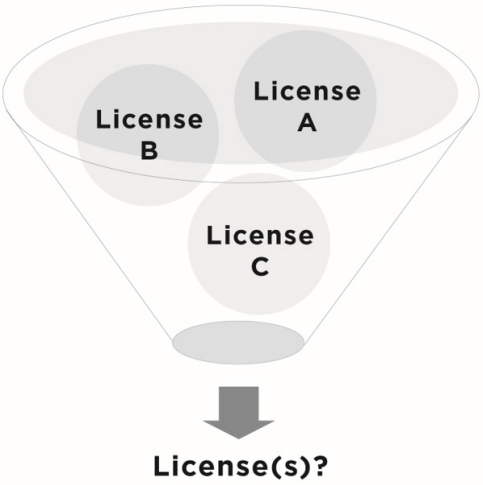


Figure 28. Combining source coming under different licenses into a single binary

License compatibility is an area where development teams need detailed guidance from Legal Counsel and should not be left to draw their own conclusions. Legal Counsel can offer such guidance via a License Compatibility Matrix that covers most popular licenses. We provide an example matrix in Table 7.

Table 7. Example license compatibility matrix (for illustration purposes only)

Compatible With:	License-A	License-B	License-C	License-D	License-E	License-F	License-G
License-A	✔				✔	✔	
License-B		✔					
License-C			✔				
License-D		✔		✔			✔
License-E					✔		
License-F			✔				
License-G	✔						✔

When development teams need to combine code under different open source licenses, they can refer to this matrix to determine if joining the software components in question creates a licensing conflict. If a license is heavily used and is not included in the matrix, the Legal Counsel would analyze the license and update the compatibility matrix accordingly.

LICENSE CLASSIFICATION

In an effort to reduce the number of questions received by Legal Counsel and to increase license and compliance process education, some companies opt to classify the most-used licenses in their products under a handful of categories. Figure 29 (next page) presents an example license classification, in which we divide the most-used licenses into four categories.

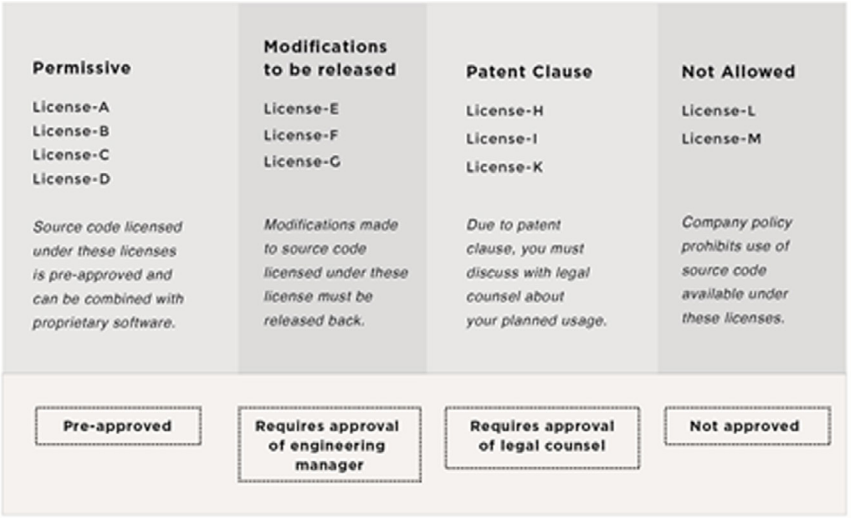


Figure 29. Example license categories (for illustration purposes only)

Pre-approved Licenses

Permissive open source licenses often fall under this category. Source code available under these licenses may be pre-approved for use by developers without having to go through the approval process with their manager and/or legal counsel. Such pre-approvals usually also require the developer to capture any notices and to make sure they are sent to the documentation team.

Licenses Requiring Manager Approval

Manager approval is required for components distributed under these licenses, since in addition to notices fulfillment (publishing license text, attribution notice, copyright notice, etc.), you have the obligation to release any source code modifications.

Licenses Requiring Legal Counsel Approval

Source code available under these licenses requires legal review and approval. This usually applies to licenses that have a patent clause.

Prohibited Licenses

Some companies flag certain licenses as “not allowed” — usage not allowed by company policy.

How can classifying licenses be helpful?

The above license categories are a way to classify licenses to make it easier for developers to know the proper course of action when integrating code under these licenses. Furthermore, it makes it easy to create an association between a license and what action should be taken and by whom. Table 8 shows one easy way developers can remind themselves of the proper actions associated with various licenses.

Table 8. A simple how-to for license classifications

Which License	Action
License A	Use with no problem
License E	Get my manager’s approval
License I	Consult with Legal
License M	Can’t use this source code
Other	Ask my manager for course of action

Please note that we provide these different examples and scenarios for illustration purposes only. You can set up a different classification model with different actions depending on your organization’s policies and guidelines.

SOFTWARE INTERACTION METHODS

As part of the compliance process, there is usually an architecture review, the goal of which is to understand how any specific software component interacts with any other software component, and the method of interaction. Architecture review should identify:

- Components that are open source (used “as is” or modified)
- Proprietary components

- Components originating from third party software providers (both open source and proprietary)
- Component dependencies
- Use of shared header files
- Component run-time context (kernel/drivers/modules, middleware, libraries, applications, etc.)
- Inter-component dependencies beyond APIs (s/w buses, IPCs, web APIs, etc.)
- Inter-language bindings

Tables 9 (below) and 10 (next page) provide additional information that Legal Counsel can provide to software developers. The tables illustrate which licenses can dynamically or statically link to which others, while respecting company policies.

Table 9. Sample dynamic linkage matrix

Can Dynamically Link To	License-A	License-B	License-C	License-D
License-A	✔	✔	✔	✔
License-B		✔		✔
License-C	✔		✔	
License-D		✔	[Requires Pre-Approval]	✔

For example, looking at Table 9, source code licensed under License-B can dynamically link to source code license under License-D. However, source code licensed under License-C cannot dynamically link to source code licensed under License-B. Also, note that linkages may not always be reciprocal between licenses.

Similarly, looking at Table 10, source code licensed under License-A can statically link to source code license under License-C. However, source code licensed under License-A cannot statically link to source code

licensed under License-B. Some linkage combination may be allowed on a case-by case basis, which is why certain combinations note "[Requires pre-approval]."

Table 10. Sample static linkage matrix

Can Statically Link	License-A	License-B	License-C	License-D
License-A	✔		✔	
License-B		✔	[Requires Pre-Approval]	
License-C	✔		✔	
License-D	[Requires Pre-Approval]			✔

In the event that the architecture review reveals any linkage issue (i.e., a static or dynamic linkage that does not follow company policy as defined in the linkage matrices), then the person responsible for driving the architecture review (usually the compliance officer) would notify the software developer responsible for that software component and request a correction.

CHECKLISTS

Most companies establish checklists and use them within their development process at every major milestone. When it comes to open source compliance, several checklists can be developed and used before committing new external open source code to the product's source code repository. One example is the following checklist, used before making source code available on an external website:

- All source code components have a corresponding compliance ticket.
- All compliance tickets have been approved by engineering and legal.
- All compliance tickets are clear of any unresolved subtasks attached to them.
- Notices for all of the software components have been sent to the Documentation team for inclusion in the product documentation.

- Legal has approved the written offer notice and overall compliance documentation.
- Source code packages have been prepared and tested to compile on a standard development machine.
- Source code provided is complete and corresponds to the binaries in the product.

Such checklists minimize the probability of error and ensure that everyone involved in open source management is aware of what needs to be done before moving to the next step in the process.

CONCLUSION

Software developers need to be educated about the licenses on the various open source components they integrate and employ. Having the Legal Counsel provide this education in a very practical way is extremely helpful, as it allows software developers to have access to documented practical advice that will help answer most of their daily legal-related questions. This practical advice usually revolves around:

- Inclusion of open source components into proprietary or third party source code or vice versa
- Linking open source components into proprietary or third party source code or vice versa
- Interaction methods between various software components (proprietary, third party, open source)
- License obligations that must be met when using open source components

Open source compliance is easy to achieve once you have built up your program, created a policy and process, established staffing, and enabled your team with tools to assist in the automation of the compliance execution.

Chapter 10

THE OPENCHAIN PROJECT

The OpenChain Project focuses on increasing open source compliance in the supply chain. This issue, often perceived solely as a legal concern or as corporate low priority, is tied to making sure that open source as useful as possible to meet business objectives. Because open source is about the use of third party code, compliance is the nexus of where equality of access, safety of use and reduction of risk can be found. OpenChain builds trust between organizations to accomplish this.

THE BUSINESS CASE FOR COMPLIANCE

Today many companies understand open source with respect to code reuse and act as major supporters of open source development. However, addressing open source license compliance in a systematic, industry-wide manner has proven to be a somewhat elusive challenge. The global IT market has not yet seen a significant reduction in the number of open source compliance issues discoverable in areas like consumer electronics over the last decade.

The majority of compliance issues originate in the sharing multiple hardware and software components between numerous legal entities. The global supply chain features participants that are simultaneously intertwined and disparate. It is possible to have companies making hardware, companies making software and companies making combined components collaborating around a relatively small part of a finished B2C device. The results in terms of products are often outstanding but the challenge of keeping track of everything is substantial.

PROCESSES ACROSS ORGANIZATIONS

Open source presents a specific challenge in the global supply chain. This is not because open source is inherently problematic but rather due to the varying degree of exposure and domain knowledge that companies possess.

By way of example, a company developing a small component that requires a device driver may have technical staff unfamiliar with open source development methodologies and licensing. One mistake, one misunderstanding, and one component deployed in dozens of devices can present an issue. Most compliance challenges arise from mistakes similar to this. Few, if any, originate with intent.

Ultimately solving open source compliance challenges involves solving open source compliance in the supply chain. This is no small task: there are thousands of companies across dozens of national borders using numerous languages. The solution lies beyond the realm of inter-company negotiation. To address open source compliance challenges the supply chain must align behind certain shared approaches.

Awareness of this fact and the provision of a practical solution are two different matters. It takes time for ideas and suggested approaches to percolate and mature. It takes input from lawyers, engineering managers, developers and political scientists. It takes, in short, a while for the ingenuity of the human community to bounce ideas back and forth until a simple, clear approach is found.

THE PLACE OF THE OPENCHAIN PROJECT

The OpenChain Project formally launched in October 2016 and is hosted by The Linux Foundation. It originated in discussions that occurred three years earlier and continued at an increasing pace until a formal project was born. The basic idea was simple: identify key recommended processes for effective open source management. The goal was equally clear: reduce bottlenecks and risk when using third-party code to make open source license compliance simple and consistent across the supply chain. The key was to pull things together in a manner that balanced comprehensiveness, broad applicability, and real-world usability.

The OpenChain Project is building an industry standard for license compliance. It can be understood as the foundation for open source compliance in the supply chain. Engagement and adoption is free of cost and supported by a vibrant community backed by leading multinationals across multiple sectors.

There are three interconnected parts to the OpenChain Project.

- A Specification that defines the core requirements of a quality compliance program.
- A Conformance method that helps organizations display adherence to these requirements.
- A Curriculum to provide basic open source processes and best practices.

DEFINING KEY REQUIREMENTS OF QUALITY OPEN SOURCE COMPLIANCE PROGRAMS

The core of the OpenChain Project is the Specification. This identifies a series of processes that help ensure organizations of any size can address open source compliance issues effectively. The main goal of organizations using the OpenChain Specification is to become conformant. This means that their organization meets the requirements of a certain version of the OpenChain Specification. A conformant organization can advertise this fact on their website and promotional material, helping to ensure that potential suppliers and customers understand and can trust their approach to open source compliance.

The Specification (available from **https://www.openchainproject.org/spec**) is split into five sections describing goals that display the adoption and use of key requirements for quality open source compliance programs. We illustrate this in Section 1.1 of Goal 1, one of three sections constituting the accomplishment of the full Goal. Each Section is accompanied by related verification material and a rationale to ensure replicability and clarity.

```
Goal 1: Know Your FOSS Responsibilities

1.1 A written FOSS policy exists that governs FOSS license compliance
of the Supplied Software distribution. The policy must be internally
communicated.
```

```
Verification Material(s):

1.1.1 A documented FOSS policy.

1.1.2 A documented procedure that makes Software Staff aware of the
existence of the FOSS policy (e.g., via training, internal wiki, or other
practical communication method).


Rationale:

To ensure steps are taken to create, record and make Software Staff aware
of the existence of a FOSS policy. Although no requirements are provided
here on what should be included in the policy, other sections may impose
requirements on the policy.
```

PROVIDING AN AVENUE TO CHECK CONFORMANCE WITH KEY PROCESSES

While the OpenChain Specification provides a clear framework for describing and confirming adherence to the key requirements of a quality open source compliance program, it is not a complete solution in and of itself. Supporting the Specification is a Conformance review that can be checked via a free online self-certification questionnaire. This is the quickest, easiest and most effective way to check and confirm adherence to the OpenChain Specification Goal by Goal. There is also a manual conformance document available for organizations whose process requires a paper review or disallows web-based submissions. Both the online and the manual conformance can be completed at a pace decided by the conforming organization and both methods remain private until a submission is completed. An illustrative example of this approach can be found in the online self-certification questions related to Section 1.1 of the OpenChain Specification:

```
1.a:    Do you have rules that govern FOSS license compliance of the
Supplied Software distribution?

1.b:    Are these rules internally communicated?

1.c:    Are these rules documented?
```

```
1.d:    Is your Software Staff aware of the rules that govern FOSS license
compliance of the Supplied Software distribution?

1.e:    Do you document, how you make your Software Staff aware of the
existing procedures that govern FOSS license compliance of the Supplied
Software distribution?

1.f:    Do you make your software staff aware of the existence of the FOSS
policy using at least one of the following methods?

        • Training

        • Internal documentation

        • Other practical communication methods?
```

If a company can answer “yes” to each of these questions then it conforms fully with Section 1.1 of the OpenChain Specification. If one or more answers are “no” then it becomes easier to identify where resources should be allocated to improve the governance of open source in the company.

The OpenChain Conformance material and online web app is available from **https://www.openchainproject.org/conformance**.

SUPPORTING CONFORMANCE WITH EDUCATIONAL MATERIAL

While the OpenChain Specification and Conformance help companies identify and confirm adherence to the key requirements of quality open source compliance programs, the OpenChain Curriculum helps organizations by providing reference process, policy and training material. It provides generic, refined and clear examples of how companies of all sizes frame their inbound, internal and outbound software compliance. The OpenChain Curriculum material is available with very few restrictions to ensure organizations can use it in as many ways as possible. To accomplish this all the core is licensed as CC-0, effectively public domain, so remixing or sharing it freely for any purpose is possible.

A good example of the OpenChain Curriculum material can be found in the general training slides to support open source compliance knowledge-sharing in companies from a wide range of sectors. These slides are

split into eight chapters that guide users from the basics of IP through to specifics about developer activities:

1. What is Intellectual Property?
2. Introduction to FOSS Licenses
3. Introduction to FOSS Compliance
4. Key Software Concepts for FOSS Review
5. Running a FOSS Review
6. End to End Compliance Management (Example Process)
7. Avoiding Compliance Pitfalls
8. Developer Guidelines

At the end of each chapter there is a short questionnaire entitled “Check Your Understanding” to confirm comprehension.

For example, at the end of chapter one the following questions are asked:

- What type of material does copyright law protect?
- What copyright rights are most important for software?
- Can software be subject to a patent?
- What rights does a patent give to the patent owner?
- If you independently develop your own software, is it possible that you will need a copyright license from a third party for that software? A patent license?

Taken together the eight chapters provide a solid foundation for engaging with open source compliance and the relevant “Check Your Understanding” questions provide a simple, short exam to confirm staff are up-to-speed.

The OpenChain Curriculum is available from **https://www.openchainproject.org/curriculum**.

ENCOURAGING ADOPTION ACROSS MULTIPLE MARKET SEGMENTS

The OpenChain Project provides a compelling approach to making open source compliance more consistent and more effective across multiple market segments. However, good ideas need implementation, and in open source, this inevitably hinges on a supporting community. Fourteen Platinum Members currently support the OpenChain Project's development and adoption: Adobe, ARM, Cisco, Comcast, GitHub, Harman, Hitachi, Qualcomm, Siemens, Sony, Toyota, Western Digital, and Wind River. There is also a wide community of almost 200 participants on the main mailing list that share and remix ideas.

At its core, the OpenChain Project is about providing a simple, clear method of building trust between organizations that rely on each other to share code and create products. Any organization that is OpenChain Conformant is aligning behind key requirements that their peers agree are required in a quality compliance program. This is about confirming overarching processes and policies, while allowing the specifics of each process and policy to be crafted by each organization to suit its specific needs.

The OpenChain Specification is at version 1.2 and is ready for adoption today by any organization that creates, uses, or distributes open source code. The online conformance is free of charge.

The mailing list and Work Team are open to everyone. This is the first unifying approach to addressing the challenge of open source compliance in the supply chain, and it has the potential to be transformative in ensuring effective allocation of resources towards open source compliance matters.

The OpenChain Project provides a quick start guide that is available from **https://www.openchainproject.org/quick-start**.

GETTING INVOLVED

Due to its open source origins, the OpenChain Project has an intentionally low barrier to entry. It hosts a series of public mailing lists, teleconferences on the first and third Monday of each month and workshops adjacent to

major open source events. All of these activities are open to participation from any party at any time.

Formally speaking the OpenChain Project consists of work teams, a steering and outreach committee, and a governing board. The work teams are accessible by the activities listed above. The steering and outreach committee is accessible to members, work team leads and one elected representative from the broader community. Finally, the governing board is accessible to Platinum Member company representatives.

All participants in the OpenChain community began their involvement via the work teams and evolved their engagement over time based on their specific requirements. This path is recommended for parties considering engagement in the future. The first step is to join the main OpenChain Project mailing list and informally collaborate with the wider community. Everyone from individuals to representatives of multinational companies are equally welcome.

To engage with the OpenChain Project community please visit **https://www.openchainproject.org/community**.

Chapter 11

SOFTWARE PACKAGE DATA EXCHANGE® (SPDX®)

INTRODUCTION

Having an accurate way to identify the software running on a system is at the heart of being able to monitor for security vulnerabilities, as well as conforming to the licenses associated with the software. The software in enterprise organizations is increasingly based off of upstream open source projects, some of which is downloaded by the organization directly, some of which has been passed through a supply chain. An accurate software bill of materials is needed in all cases.

The Software Package Data Exchange® (SPDX®) is an open standard for communicating software bill of material information between organizations as well as from upstream open source projects into an organization. A software bill of material (SBOM) needs to be precise and unambiguous in order to accurately identify the code being used in products, security vulnerabilities associated with that code, the license obligations and those who can be negotiated with to change the license obligations (copyright holders). When we have a common language to communicate these concepts, information can be effectively shared, and does not need to be regenerated at each step in the supply chain.

A common language and vocabulary to express security, licensing, and copyright information for products, components, packages, files and code snippets, enable tools to be created and facilitate the introduction of automation. The SPDX project was started to provide such a language for summarizing SBOM information so that it could be exchanged. As the initial version of the specification was being developed, it was recognized that there was a need for a common list of licenses represented by unique identifiers. Scanning software to determine the appropriate license to use was error prone due to the natural language syntax that is used in license

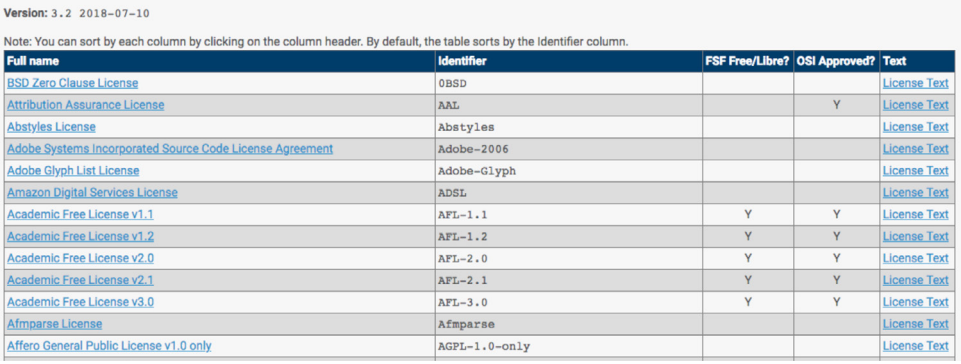
headers, and the variations that have emerged. What is easy for a human to understand is not always understandable for source code scanning tool. In recent years, the project has also adopted a syntax for using the license identifiers in source code to clarify which license should be apply and fixing the problem at the source level as well.

The SPDX specification and license list (**http://www.spdx.org**) were created by developers, supply chain, security and legal professionals collaborating. The interdisciplinary team has been incrementally refining the SPDX specification (currently at 2.1.1 [**https://spdx.github.io/spdx-spec/**]) and the list of recognized licenses (currently at version 3.2) (**https://spdx.org/licenses/**) since 2010. If there is a use case you're not sure how to represent with the specification, you are encouraged to reach out to the volunteers at **spdx-outreach@spdx.org** and ask about it. If someone can't figure out a solution, the use case will be added to the topics for the specification team to address. The project also has a set of Java (**https://github.com/spdx/tools/**) and Python **(https://github.com/spdx/tools-python)** based tools to help with validation of documents, and conversion between the supported formats.

SPDX License List

The SPDX license list identifiers (**https://spdx.org/spdx-license-list/request-new-license**) have been widely adopted at this point and the identifiers are recognized by an increasing number of upstream open source projects, companies, organizations, governments, and tool vendors. The purpose of the SPDX License List is to enable easy and efficient identification of such licenses and exceptions in an SPDX document, in source files or elsewhere. Use of this standard streamlines license identification across the supply chain while reducing redundant work.

The SPDX License List is a list of commonly found licenses and exceptions used in free and open source and other collaborative software or documentation. The SPDX License List includes a standardized short identifier, full name, vetted license text including matching guidelines markup as appropriate, and a canonical permanent URL for each license and exception. When you go to the SPDX License List web site, you'll see the SPDX License List table.



Version: 3.2 2018-07-10

Note: You can sort by each column by clicking on the column header. By default, the table sorts by the Identifier column.

Full name	Identifier	FSF Free/Libre?	OSI Approved?	Text
BSD Zero Clause License	0BSD			License Text
Attribution Assurance License	AAL		Y	License Text
Abstyles License	Abstyles			License Text
Adobe Systems Incorporated Source Code License Agreement	Adobe-2006			License Text
Adobe Glyph List License	Adobe-Glyph			License Text
Amazon Digital Services License	ADSL			License Text
Academic Free License v1.1	AFL-1.1	Y	Y	License Text
Academic Free License v1.2	AFL-1.2	Y	Y	License Text
Academic Free License v2.0	AFL-2.0	Y	Y	License Text
Academic Free License v2.1	AFL-2.1	Y	Y	License Text
Academic Free License v3.0	AFL-3.0	Y	Y	License Text
Afmparse License	Afmparse			License Text
Affero General Public License v1.0 only	AGPL-1.0-only			License Text

The first thing to note is the SPDX License List version number. It is important to keep in mind that this is a living list, and gets updates approximately every quarter. If you don't see a license you are commonly encountering in open source code, please feel free to send a request for proposing a license or an exception to be added to the SPDX License List (**https://spdx.org/spdx-license-list/request-new-license**).

The SPDX license list can also be programmatically accessed as well, so that your organization's tools can use the license text and matching guidelines as well. The recommended way to get programmatic access to the latest version of the license list is through the project on GitHub (**https://github.com/spdx/license-list-data**), rather than scraping the web site. The repository there contains various generated data formats for the SPDX License List, including JSON, RDFa/HTML, RDF NT, RDF turtle, RDF/XML, HTML as well as a simple text version. More details on how to programmatically access the SPDX license list can be found on the GitHub repo (**https://github.com/spdx/license-list-data/blob/master/accessingLicenses.md**) as well.

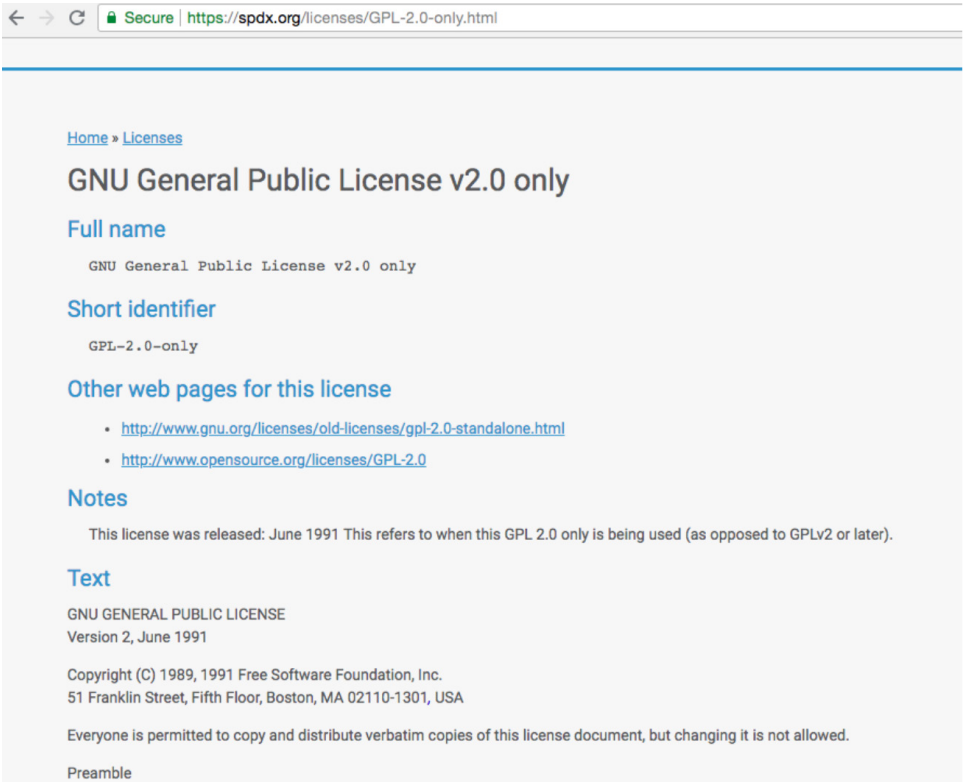
In the SPDX License List table, you'll see the columns for:

- **Full Name** of the license.
- **Identifier** for the license. This "short identifier" also gets referred to as the SPDX-id in some places.
- **FSF Free/Libre?** If the license is considered free by the FSF

(**https://www.gnu.org/licenses/license-list.en.html**), this field will indicate "Y" and is otherwise left blank

- **Is OSI Approved?** If the license is OSI-approved, this field will indicate "Y" and otherwise left blank
- **License Text** of the license. The full text of the license is provided as well as any standard headers associated with a license.

If you click on the column headers it will sort the SPDX License List table by those fields. By clicking on the "Full Name" or the "License Text" you'll also be taken to a canonical permanent URL for that license that provides more information about the license. An example of the GPL-2.0-only license page is reproduced below. The permanent URL is made by appending the short identifier to the "**https://spdx.org/licenses/**" prefix.



On this page, in addition to the fields shown in the SPDX License List table are:

- **Other web pages for this license.** This Includes URL for the official text of the license or exception and f the license is OSI approved, also include URL for OSI license page
- **Notes.** Factual information may be noted here, including links to translations.
- **Standard Header.** Use when the license indicates specific text intended to be put in the header of source files.

SPDX License IDs

Accurately identifying the license for open source software is important for license compliance. However, determining the license can sometimes be difficult due to a lack of information or ambiguous information. Even when there is some licensing information present, a lack of consistent ways of expressing the license can make automating the task of license detection very difficult, thus requiring significant amounts of manual human effort. There are some commercial tools applying machine learning to this problem to reduce the false positives, and train the license scanners, but a better solution is to fix the problem at the upstream source.

In 2013, the U-boot project decided to use (**https://git.denx.de/?p=u-boot.git;a=commit;h=eca3aeb352c964bdb28b8e191d6326370245e03f**) the SPDX license identifiers in each source file instead of boilerplate that had been used up to that point. The initial commit message had an eloquent explanation of reasons behind this transition.

```
Licenses: introduce SPDX Unique License Identifiers

Like many other projects, U-Boot has a tradition of including big blocks
of License headers in all files. This not only blows up the source code
with mostly redundant information, but also makes it very difficult to generate
License Clearing Reports. An additional problem is that even the same licenses
are referred to by a number of slightly varying text blocks (full, abbreviated,
different indentation, line wrapping and/or white space, with obsolete
address information, ...) which makes automatic processing a nightmare.

To make this easier, such license headers in the source files will be
replaced with a single line reference to Unique License Identifiers as
```

```
defined by the Linux Foundation's SPDX project [1]. For example, in a
source file the full "GPL v2.0 or later" header text will be replaced by a
single line:

    SPDX-License-Identifier:    GPL-2.0+

We use the SPDX Unique License Identifiers here; these are available at
[2].

...

[1] http://spdx.org/

[2] http://spdx.org/licenses/
```

The SPDX project liked the simplicity of this approach, and formally adopted the syntax for embedding SPDX-License-Identifier's into the project and documented the syntax in SPDX specification version 2.1 "Appendix V: Using SPDX short identifiers in Source Files". Since then, other upstream open source projects and repositories have adopted use of these short identifiers to identify the licenses in use, including github in its licenses-API.

In 2017, the Free Software Foundation Europe created a project called REUSE.software that provided guidance for open source projects on how to apply the SPDX-License-Identifiers into projects. The REUSE.software guidelines were followed for adding SPDX-License-Identifiers into the Linux kernel, later that year.

The SPDX-License-Identifier syntax used with short identifiers from the SPDX License List (referred to as SPDXLIDs) can be used to indicate relevant license information at any level, from package to the source code file level. The "SPDX-License-Identifier" phrase and a license expression formed of SPDXLIDs in a comment form a precise, concise and language neutral way to document the licensing that is simple to machine process. This leads to source code that is easier to read, which appeals to developers, as well as enabling the licensing information to travel with the source code.

To use SPDXLIDs in your project's source code, just add a single line in the following format, tailored to your license(s) and the comment style for that file's language. For example:

```
// SPDX-License-Identifier: MIT

/* SPDX-License-Identifier: MIT OR Apache-2.0 */

# SPDX-License-Identifier: GPL-2.0-or-later
```

To learn more about how to use SPDXLIDs with your source code, please see the documentation in the SPDX project, REUSE.software and David Wheeler's tutorial (**https://github.com/david-a-wheeler/spdx-tutorial**).

In addition to U-boot, Linux transitioning to use the SPDXLL IDs, newer projects like Zephyr and Hyperledger fabric have adopted them right from the start as a best practice. Indeed, to achieve the Core Infrastructure Initiative's gold badge, each file in the source code must have a license, and the recommended way is to use an SPDXLL ID.

The project MUST include a license statement in each source file. This MAY be done by including the following inside a comment near the beginning of each file: SPDX-License-Identifier: [SPDX license expression for project].

When SPDXLIDs are used, gathering license information across your project files can start to become as easy as running grep. If a source file gets reused in a different package, the license information travels with the source, reducing the risk of license identification errors, and making license compliance in the recipient project easier. By using SPDXLIDs in license expressions, the meaning of license combinations is understood more accurately. Saying "this file is MPL/MIT" is ambiguous, and leaves recipients unclear about their compliance requirements. Saying "MPL-2.0 AND MIT" or "MPL-2.0 OR MIT" specifies precisely whether the licensee must comply with either licenses, or either license, when redistributing the file.

As illustrated by the transition underway in the Linux kernel, SPDXLIDs can be adopted gradually. You can start by adding SPDXLIDs to new files without changing anything already present in your codebase.

SPDX Specification – Background

The Software Package Data Exchange® (SPDX®) specification is a common language for communicating the components, licenses, security information and copyrights associated with software packages. SPDX reduces redundant

work by providing a syntax and vocabulary for companies and communities to share important data, thereby streamlining and improving compliance.

Software development teams across the globe use the same open source packages, but in 2010, there was little infrastructure exists to facilitate collaboration on the analysis or share the results of these analysis activities. As a result, many groups were performing the same work leading to duplicated efforts and redundant information. The SPDX project was formed to create a data exchange format so that information about software packages and related content may be collected and shared in a common format with the goal of saving time and improving data accuracy. The specification is a living document, as new use-cases are examined, it evolves. Development progresses through collaboration between technical, business and legal professionals from a range of organizations to create a standard that addresses the needs of various participants in the software supply chain.

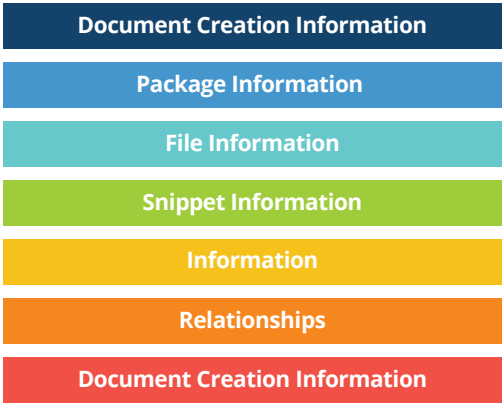
Companies and organizations (collectively “Organizations”) are widely using and reusing open source and other software packages. Accurate identification of the software is key to understanding if it contains any security vulnerability.

Compliance with the associated licenses requires a set of analysis activities and due diligence that each Organization performs independently, which may include a manual and/or automated scan of software and identification of associated licenses followed by manual verification.

The SPDX specification is developed by community members participating the SPDX project, which is hosted by the Linux Foundation. This grass-roots effort has had participation over the years from a wide variety of software developers, systems and tool vendors, foundations and the legal community—all committed to creating a common format for products, components and software packages to be able to exchange Software Bill of Materials (SBOM) data efficiently and effectively.

Overview of an SPDX Document

The SPDX specification describes the necessary sections and fields to produce a valid SPDX document.



Each SPDX document can be composed from the following:

- **Document Creation Information:** One instance is required for each SPDX file produced. It provides the necessary information for forward and backward compatibility for processing tools (version numbers, license for data, authors, etc.)
- **Package Information:** A package in an SPDX document can be used to describe a product, container, component, packaged upstream project sources, contents of a tar ball, etc. It's just a way of grouping together items that share some common context. It is not necessary to have a package wrapping a set of files.
- **File Information:** A file's important meta information, including its name, checksum licenses and copyright, is summarized here.
- **Snippet Information:** Snippets can optionally be used when a file is known to have some content that has been included from another original source. They are useful for denoting when part of a file may have been originally created under another license.
- **Other Licensing Information:** The SPDX license list does not represent all licenses that can be found in files, so this section provides a way to summarize other license that may be present in software being described.
- **Relationships:** Most of the different ways that SPDX documents,

packages, files can be related to each other can be described with these relationships.

- **Annotations:** Annotations are usually created when someone reviews the SPDX document and wants to pass on information from their review. However, if the SPDX document author wants to store extra information that doesn't fit into the other categories, this mechanism can be used.

It's important to note that not all of these sections are required in every document. The only one that is mandatory is to have a "Document Creation Information" section for each document. Then it's a matter of using the sections (and subset of the fields in each section) that describe the software and metadata information you're planning to share. However, for each package and file section that is used, there does need to be a hash, so recipients of the SPDX document can independently check if the metadata information is still valid.

Each document is capable of being represented by a full data model implementation and identifier syntax. This permits exchange between data output formats (RDFa, tag:value, spreadsheet), and formal validation of the correctness of the SPDX document. In the SPDX specification 2.2 release, the additional output formats of JSON, YAML and XML are planned to be supported. Further information on the data model can be found in Appendix III of the SPDX Specification (**https://spdx.github.io/spdx-spec/appendix-III-RDF-data-model-implementation-and-identifier-syntax/**) and on the SPDX web site (**https://spdx.org/rdf/terms/**).

Document Creation Information

There must be a "Document Creation Information" section for each SPDX document. In it, seven of the fields are required to be filled out. The version of the SPDX specification used to generate the document is the first field, as it provides the key to understand which fields are in each document. Each SPDX document is considered to be under CC0-1.0 (**https://spdx.org/licenses/CC0-1.0.html**) license, and this is denoted by the data license. Other mandatory elements are the self-identification of the document and its namespace, who created the document, and when.

Each field has a specific grammar associated with it and rules for parsing. Details of each field, rationale for the field, and parsing guidance can be found in **https://spdx.github.io/spdx-spec/2-document-creation-information/**

	Mandatory	Added	Field Name	Comment
Document Creation Information	X	1.0	2.1 SPDX Version	which version of SPDX?
	X	1.0	2.2 Data License	data in document: CC0-1.0
	X	2.0	2.3 SPDX Identifier	id of the document itself
	X	2.0	2.4 Document Name	
	X	2.0	2.5 SPDX Document Namespace	URI
		2.0	2.6 External Document Reference	
		1.2	2.7 License List Version	when document created.
	X	1.0	2.8 Creator	how was the file created? • Manual review (who, when) • Tool (id, version, when)
	X	1.0	2.9 Created	when?
		1.0	2.10 Creator Comment	comments on creator?
		1.1	2.11 Document Comment	comments on this document?

An example of this section expressed as tag:value is:

```
SPDXVersion: SPDX-2.1

DataLicense: CC0-1.0

SPDXID: SPDXRef-DOCUMENT

DocumentName: SPDX document for Time version 1.7

DocumentNamespace: http://spdx.org/documents/d3e9fef0-00a0-4b39-bb28-ff3dc75c7200

LicenseListVersion: 2.5

Creator: Tool: Source Auditor Open Source Console

Creator: Organization: Source Auditor Inc.

Created: 2018-09-26T11:44:51Z
```

Package Information

If there is a grouping of elements to be described, then a package section should be created. This section can be used to represent a product, a container, an upstream project source repository, or even an archive. It has relationships to other package information sections or file information

sections. If there are no files associated with this package in the document, then “Files Analyzed” should be set to indicate this. By using “External Reference” field, the package can be linked to security information as well as to public repositories, in addition to any “Package Download Location” provided.

There are three mandatory fields associated with describing licensing of the package. The “Concluded License”, is filled in by the creator after looking at “All License Information from Package” and “Declared License” information. As an example, the Linux kernel would have a “Conclude License” of “GPL-2.0-only with Linux-syscall-note”, the “Declared License” would be based on the contents of the COPYING FILE and “All License Information from Package” would probably have about 80 licenses listed, based on the version of the kernel selected.

Details of each field, rationale for the field, and parsing guidance can be found in **https://spdx.github.io/spdx-spec/3-package-information/**

	Mandatory	Added	Field Name	Comment
Package Information	X	1.0	3.1 Package Name	formal name by originator
	X	2.0	3.2 Package SPDX Identifier	unique ID
		1.0	3.3 Package Version	
		1.0	3.4 Package File Name	actual file name for package
		1.0	3.5 Package Supplier	
		1.0	3.6 Package Originator	
	X	1.0	3.7 Package Download Location	download URL
		2.1	3.8 Files Analyzed	files associated with package?
	X	1.0	3.9 Package Verification Code	special algorithm
		1.0	3.10 Package Checksum	
		1.2	3.11 Package Home Page	project homepage
		1.0	3.12 Source Information	
	X	1.0	3.13 Concluded License	
	X	1.0	3.14 All Licenses Information from Package	
	X	1.0	3.15 Declared License	
		1.0	3.16 Comments on License	
	X	1.0	3.17 Copyright Text	any copyrights declared?
		1.0	3.18 Package Summary Description	
		1.0	3.11 Package Detailed Description	
		2.0	3.12 Package Comment	
		2.1	3.13 External Reference	
		2.1	3.14 External Reference Comment	

An example of a package expressed as tag:value is:

```
PackageName: GNU Time

SPDXID: SPDXRef-1
```

```
PackageVersion: 1.7

PackageFileName: time-1.7.tar.gz

PackageSupplier: Organization: GNU

PackageOriginator: Organization: GNU

PackageDownloadLocation: ftp://ftp.gnu.org/gnu/time/

PackageVerificationCode: dd5cf0b17bfef4284c6c22471b277de7beac407c

PackageChecksum: SHA1: dde0c28c7426960736933f3e763320680356cc6a

PackageLicenseConcluded: GPL-2.0+

PackageLicenseInfoFromFiles: GPL-2.0+

PackageLicenseInfoFromFiles: MIT

PackageLicenseInfoFromFiles: GPL-2.0

PackageLicenseDeclared: GPL-2.0+

PackageCopyrightText: <text>Copyright (C) 1990, 91, 92, 93, 96 Free
Software Foundation, Inc.</text>

PackageSummary: <text>The 'time' command runs another program, then
displays information about the resources used by that program, collected
by the system while the program was running.</text>

PackageDescription: <text>The 'time' command runs another program, then
displays information about the resources used by that program, collected by
the system while the program was running. You can select which information
is reported and the format in which it is shown, or have `time' save the
information in a file instead of displaying it on the screen.</text>
```

File Information

Each individual file to be summarized, must have a name and a checksum associated with it. If there is any "License Information in File", then it should be documented either by an SPDX ID or via a "LicenseRef-" (see Other Licensing Information).

In some cases, the information found in the file may not be the "Concluded License" for that file, and so a second mandatory field is provided, so the license that governs the file can be made explicit. If there is any copyright information in the file it should also be summarized.

	Mandatory	Added	Field Name	Comment
File Information	X	1.0	4.1 File Name	what is name of file
	X	2.0	4.2 File SPDX Identifier	unique ID
		1.0	4.3 File Type	source, binary, ...
	X	1.0	4.4 File Checksum	SHA1, MD5, SHA256
	X	1.0	4.5 Concluded License	by SPDX document creator
	X	1.0	4.6 License Information in File	detected by scanning file
		1.0	4.7 Comments on License	
	X	1.0	4.8 Copyright Text	
		1.0	4.9 Artifact of Project Name	deprecated
		1.0	4.10 Artifact of Project Homepage	deprecated
		1.0	4.11 Artifact of Project URI	deprecated
		1.1	4.12 File Comment	
		1.2	4.13 File Notice	if Notice found in file
		1.2	4.14 File Contributor	if Contributor info in file
		1.2	4.15 File Dependencies	deprecated

In the above table, some fields are marked as deprecated and should not be used, however were present in prior versions of this section. Details of each field, rationale for the field, and parsing guidance can be found in **https://spdx.github.io/spdx-spec/4-file-information/**.

An example of a package expressed as tag:value is:

```
FileName: ./time.c

SPDXID: SPDXRef-4

FileType: SOURCE

FileChecksum: SHA1: 712d7f9dfde674283596ae2088550e3ff23ae1ba

LicenseConcluded: GPL-2.0+

LicenseInfoInFile: NOASSERTION

FileCopyrightText: <text>Copyright Free Software Foundation, Inc</text>
```

Snippet Information

Each instance of “Snippet Information” needs to be associated with a specific “File Information” in an SPDX Document via the File’s “SPDX Identifier”. The “Snippet Byte Range” field is used identify the part of the file being described. The “Snippet Concluded License” and any “Snippet Copyright Text” are also required to be documented when a snippet section is used.

	Mandatory	Added	Field Name	Comment
Snippet Information	X	2.1	5.1 Snipper SPDX Identifier	unique ID
	X	2.1	5.2 Snipper from File SPDX Identifier	unique ID
	X	2.1	5.3 Snippet Byte Range	number:number
		2.1	5.4 Snippet Line Range	number:number
	X	2.1	5.5 Snippet Concluded License	By SPDX document creator
		2.1	5.6 License Information in Snippet	detected by scanning file
		2.1	5.7 Snippet Comments on License	
	X	2.1	5.8 Snippet Copyright Text	
		2.1	5.9 Snippet Comments	
		2.1	5.10 Snippet Name	for convenience

Details of each field, rationale for the field, and parsing guidance can be found in **https://spdx.github.io/spdx-spec/5-snippet-information/**.

An example of a snippet expressed as tag:value is:

```
SnippetSPDXID: SPDXRef-5

SnippetFromFileSPDXID: SPDXRef-2

SnippetByteRange: 889:9002

SnippetLineRange: 24:245

SnippetLicenseConcluded: Apache-2.0

LicenseInfoInSnippet: BSD-2-Clause-FreeBSD

SnippetCopyrightText: <text>Copyright 2001-2016 The Apache Software
Foundation</text>

SnippetComment: <text> This snippet should have a related package with
an external referenced, however, the maven-plugin only supports external
references for the main package </text>

SnippetName: Apache Commons Math v. 3.6.1
```

Other Licensing Information

One instance of “Other Licensing Information” should be created for every unique license or licensing information reference detected in the files or packages described in the document that does not match one of the licenses on the SPDX License List. Each found license documented must have a “License Identifier” assigned to the verbatim “Extracted Text” found. The “License Identifier” is required to start with the prefix “LicenseRef-”

to help identify it in the rest of the document. In some case the extracted license may have a formal name in other contexts, and the "License Name" is an optional field to permit recording this if known. Details of each field, rationale for the field, and parsing guidance can be found in **https://spdx.github.io/spdx-spec/6-other-licensing-information-detected/**.

	Mandatory	Added	Field Name	Comment
Other Licensing Information*	X	1.0	6.1 License Identifier	LicenseRef-uniqueID
	X	1.0	6.2 Extracted Text	text found during scans
		1.1	6.3 License Name	formal name
		1.1	6.4 License Cross Reference	text found during scans
		1.1	6.5 License Comment	unique ID

*** OPTIONAL** NOTES: • Provides a way to identify licenses not on the SPDX License List • SPDX aims for ~90% coverage with short forms license identifiers - NOT exhaustive · Although there are a lot of licenses "in the wild," a smaller number covers most project

An example of an extracted license expressed as tag:value is:

```
LicenseID: LicenseRef-FaustProprietary

ExtractedText: <text>FAUST, INC. PROPRIETARY LICENSE:

FAUST, INC. grants you a non-exclusive right to use, modify, and distribute
the file provided that (a) you distribute all copies and/or modifications
of this file, whether in source or binary form, under the same license,
and (b) you hereby irrevocably transfer and assign the ownership of your
soul to Faust, Inc. In the event the fair market value of your soul is
less than $100 US, you agree to compensate Faust, Inc. for the difference.
Copyright (C) 2016 Faust Inc. All, and I mean ALL, rights are reserved.</
text>

LicenseName: Faust (really) Proprietary License

LicenseComment: <text>This license was extracted from the file
InsufficientKarmaException</text>
```

Relationships

This field can be used to provide information about the relationship between two SPDX elements. For example, you can represent a relationship between two different Files, between a Package and a File, between two Packages, or between one SPDXDocument and another SPDXDocument.

	Mandatory	Added	Field Name	Comment
Relationships*	X	2.0	7.1 Relationship	unique ID
		2.0	7.2 Relationship Comment	text found during scans

*** OPTIONAL**

The relationships between two elements that are supported are:

- DESCRIBES, DESCRIBED_BY
- CONTAINS, CONTAINED_BY
- GENERATES, GENERATED_FROM
- ANCESTOR_OF, DESCENDANT_OF
- VARIANT_OF, COPY_OF
- DISTRIBUTION_ARTIFACT, PATCH_FOR, PATCH_APPLIED
- FILE_ADDED, FILE_DELETED, FILE_MODIFIED
- EXPANDED_FROM_ARCHIVE
- DYNAMIC_LINK, STATIC_LINK
- DATA_FILE_OF, TEST_CASE_OF, BUILD_TOOL_OF, DOCUMENTATION_OF
- OPTIONAL_COMPONENT_OF, METAFILE_OF, PACKAGE_OF
- AMENDS
- PREREQUISITE_FOR, HAS_PREREQUISITE
- OTHER

This set of relationships was determined by examining common use cases in the supply chain. Others can be added if a use case can be shown not to be able to be represented with the current set by opening a new issue (**https://github.com/spdx/spdx-spec/issues**) against the SPDX specification.

A detailed description and examples of each relationship can be found in **https://spdx.github.io/spdx-spec/7-relationships-between-SPDX-elements/**.

A Relationship would follow a file or package section, and may have a comment associated with it:

```
Relationship: SPDXRef-2 PREREQUISITE_FOR SPDXRef-1

RelationshipComment: <text>The package foo.tgz is a pre-requisite for
building the executable bar.</text>
```

Annotations

This section permits a person, organization or tool to add comments about elements in an SPDX document. Comments can be made on file, package, or the entire document. Annotations are usually created when someone reviews the file, but if an author wants to store extra information about one of the elements during creation this can be used as well. If an annotation is to be made, all the sections need to be filled out. More details on the fields and values can be found in: **https://spdx.github.io/spdx-spec/8-annotations/**.

	Mandatory	Added	Field Name	Comment
Annotations*	X	2.0	8.1 Annotator	the person, company, or tool which provided the annotation
	X	2.0	8.2 Annotation Date	
	X	2.0	8.3 Annotation Type	reviewer or other
	X	2.0	8.4 SPDX Identifier Reference	unique ID
	X	2.0	8.5 Annotation Comment	free form information

*** OPTIONAL** NOTES: • Annotations with type = reviewer cover all functionality deprecated Reviewer Information (section 9).

An example annotation could look like:

```
Annotator: Person: John Smith

AnnotationDate: 2018-01-29T18:30:22Z

AnnotationType: REVIEW

SPDXREF: SPDXRef-5

AnnotationComment: <text>Copyright on snippet should be Copyright 2010-
2012 CS Systèmes d'Information </text>
```

Tools and Other Resources for Sharing SPDX Documents

Many companies generate SPDX documents from their internal infrastructure now. Wind River was one of the first to make SPDX documents the standard output format for SBOMs associated with their products. Other companies including TI, Intel, ARM, Samsung have been public about their use of the information in internal databases.

Tools That Can Generate SPDX Documents

FOSSology

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.

- Source code for the tool is available from: **https://github.com/fossology/fossology**
- Installation instructions are available from: **https://wiki.fossology.org/handson**

ScanCode

ScanCode is an open source tool to scan code and detect licenses, copyrights, packages metadata & dependencies and more, to find, discover, and inventory open source and third-party components used in your code.

Source code and installation instructions are available from **https://github.com/nexB/scancode-toolkit**

FOSSID

Commercial tool able to create SPDX documents. **http://fossid.com/services/**

Black Duck Hub

Commercial tool able to export SPDX documents. **https://www.blackducksoftware.com/solutions/open-source-license-compliance**.

Source Auditor

Commercial tool able to create SPDX documents. **http://sourceauditor.com/blog/source-auditor-supports-spdx-2/**

Protecode

Commercial tool able to create SPDX documents.
http://www.protecode.com/license-compliance-is-evolving-with-spdx/

Tools Able to Import SPDX Documents

SPDXTools

These tools are able to consume SPDX documents, and verify they are correct. The tools also include translators between formats, as well as plugins and summarizers.

SPDXTools are available from **https://github.com/spdx/tools**

FOSSology

After version 3.2, FOSSology is able to import SPDX RDF documents, and include the information in its database. Please refer to the 3.2.0 release note for details: **https://github.com/fossology/fossology/releases/tag/3.2.0**.

If you don't see your favorite commercial tool on this list, please reach out to that vendor and let them know you want the ability to import SPDX documents.

Help Improve SPDX

The SPDX community consists of individuals and companies who are producing and consuming SPDX documents, as well as those who contribute to the SPDX specification, License List and tools.

There are three SPDX teams that work on different aspects of the SPDX project. They are:

Technical (**https://spdx.org/WorkgroupTechnical**) - Maintains and publishes the specification and tools.

Outreach (**https://spdx.org/WorkgroupOutreach**)– Supports SPDX adoption efforts.

Legal (**https://spdx.org/legal-team**) - Publishes and maintains the license list and associated collateral.

Each team has its own meetings and mailing list and in general there are conference calls at least bi-weekly with the technical team meeting every week. There is also a general mailing list (**https://lists.spdx.org/g/spdx**) along with a monthly meeting for everyone in the SPDX community, the general mailing list is low volume and provides an overview of the workgroups.

Chapter 12

EVALUATING SOURCE CODE SCANNING TOOLS

There are a number of companies providing open source compliance tools and services. The question of what tool is best for a specific usage model and environment always comes up. Hence, this chapter that presents a number of metrics that we recommend you consider when evaluating multiple source code scanning and identification tools.

KNOWLEDGE BASE

Size of the Knowledge Base

This database stores information about open source software. The larger the database is the more open source code you will be able to identify.

Frequency of Updates to the Knowledge Base

Compliance service and tools providers update their databases on a regular basis. Some companies do the update three or four times per year, other companies do it on at much higher frequency (up to daily). Ideally, you would want to have the largest and most updated database to increase your chances identifying newly created open source code.

DETECTION CAPABILITIES

Ability to Identify Source Code Snippets

This capability is one of the most critical features that a source code scanner should have. When developers copy open source code into your code base, they do so in two ways: whole components and snippets. Copying a whole component is similar to downloading for example zlib and adding it into your internal git repository, compiling it, and linking other code into the produced zlib library. Now, the hard problem here is to

identify source code snippets that developers copied from an open source component into proprietary or third party code.

Let us assume that a developer has copied 100 lines of code from zlib (just to continue with the same example) into your proprietary component. Here are some questions that will arise:

- Is your source code scanning engine capable of identifying those 100 lines and pinpointing their origin and license?
- Is your engine capable of identifying the true original source of those 100 lines? As you know, hundreds of open source components use zlib code and integrate it in their code. Therefore, when you scan source code containing snippets originating from zlib, those 100 lines will have hundreds of matches. If your scanning engine is capable of pinpointing the original source (with original license) on the top list (#1) of the match list, then you are gold. Many source code scanning engines do not identify snippets, and among those who do, very few are able to execute such a scenario.

Ability to Auto-Identify Source Code (Components and Snippets)

Most of the source code scanning engines, especially those with snippet support, do generate a significant amount of false positives that need to be investigated and resolved manually leading to endless hours of manual labor. This is an ongoing problem with some of the most known products in the market today. When evaluating such products, we recommend prioritizing scanning engines capable of auto-identifying source code snippets leading to the least amount of false positives that you need to vet manually.

EASE OF USE

Ease of use is important because if all your engineers have access and use the scanning tool (versus only compliance engineers), you may avoid compliance problems way before these arise and before engineers integrated the new code with your build system.

You would want an easy to use tool that minimizes the learning curve and avoid the need for costly professional training.

OPERATIONAL CAPABILITIES

Use for M&A Purposes

Many of the compliance tool vendors impose limitations via their licensing agreement on your ability to use the tool in scenarios outside just scanning code in relation to ongoing development efforts. You need to be aware of this fact and make sure that you are able to use the tool, for instance, for any M&A transaction your company is considering.

Support for Different Audit Models

There are three audit models (discussed in the following chapter): traditional, blind and DIY. All companies support the traditional model. Very few support the DIY. Only one supports the blind audit model, which provides the most secure, and private auditing model in M&A scenarios.

Programming Languages Agnostic

Some tools are, by admission of their creators, very good working with specific programming languages, and not so with others. This is interesting, as you would expect any scanning and identification engine to be agnostic to programming languages. Most tools are not; very few are agnostic to languages.

Speed of Source Code Scans

Speed of source code scans is a pain point for many products on the market today. For instance, one specific company designed and developed their own database that is perfectly suitable for manipulating the type of such data. As a result, they have lightning fast scans that are exponentially faster than other existing tools. Furthermore, the speed of scans is particularly useful when you integrate the scanning tool with your continuous integration process.

INTEGRATION CAPABILITIES

Support for APIs and a CLI

Using a scanning tool is not limited to a UI-based usage. Ideally, companies want to integrate the tool with their existing development and build systems and processes. Such a scenario is doable if the scanning tool supports APIs and a CLI that would allow system administrators from interacting with the tool outside the UI.

SECURITY VULNERABILITY DETECTION CAPABILITIES

Size of the Security Vulnerabilities Database

This database contains information about known security vulnerabilites that enable the tool to detect security related problems in the source code. Please note the use of ”source code” and not specifically open source code in the previous sentence. The reason being that developers may copy code snippets from open source components into proprietary or third party components. If the copied code contained a known security vulnerability, then when you scan the proprietraty component, your engine should be able to flag the vulnerability.

Frequency of Updates to Security Vulnerability Database

Service providers update their databases on a regular basis. The more frequent the update cycle, the better it is in term or finding vulnerabilities as soon as they have been identified.

Sources of Security Vulnerability Information

Multiple sources can be used to populate the database of security vulnerabilities in open source components. When evaluating compliance tools that offer this service, we recommend you investigate this aspect and explore the actual mechanics of the updates and the various sources (direct and indirect) used to collect information on security vulnerabilities and on which basis recommendations are presented to fix those vulnerabilities.

Support for Advanced Vulnerability Discovery

This specific feature is not compliance related. However, since most compliance tools have access to source code, vendors started integrating the ability to flag security vulnerabilities.

There are two methods to do the discovery of security vulnerabilities: Traditional and smart (author's own classification). Following the traditional method, the scanning engine discovers that you are using zlib 1.2.11 (fictional example) and at the same time, the engine is aware that this specific version of zlib has a security vulnerability. At this point, the user of the tool is notified via a visual warning of some sort (depends on the specific tool) that the tool has discovered a vulnerability and in some cases the tool will offer a recommendation on how to fix it (upgrade version or a patch is now available).

This is a good start but it turns a blind eye the most common use of open source software: snippets. The second most interesting method, the smart method, works at snippet level to discover security vulnerabilities. Let us examine this scenario. A developer copies 100 lines from zlib 1.2.11 (again fictional example) into your proprietary source code base.

You ran the scan and the tool identifies the 100 lines to be originating from zlib 1.2.11 and flags that those lines contain a vulnerability, with pointer to additional information and suggestions to resolve the issue.

The smart method is also easier to automate in a development process, as you actually know that you use vulnerable code. If a scanning engine does not support snippet discovery, then it is extremely hard, if not impossible for it to identify security vulnerability copied from one body of code into another.

COST

Cost is a vague metric as most focus in put on the yearly licensing cost of the tool, while there are many other hidden costs to evaluate that we explore next.

Infrastructure Cost

IT infrastructure costs related to hosting the solution or using it via cloud. It involves the usage of servers that customers need to buy, set up and maintain, including the cost to upgrade that infrastructure and depending on its size, the cost of a dedicated system administaror.

Operational Cost

Cost related to managing the results that the tool provides. That involves inspecting and interpreting the results and take according action. A tool that does auto identification of false positives will lower the cost associated with labor needed to manually identify those thousands of false positives.

Licensing Cost

The cost of the yearly software license for using the tool (cost per seat, unlimited seats), cost to access to SDK so you can integrate your internal tools with the scanning engine, and possibly the cost of any private customization that you want to introduce to fit your needs.

Integration Cost

Integration costs are hard to estimate but they typically evolve about the ability to integrate the tool into your workflows and processes with minimal disruptions.

Lock-in Cost

Companies often ignore or do not pay enough attention to the lock-in factor and costs associated with building the whole compliance environment around a specific tool. When choosing a new tool, we recommend putting enough consideration in this aspect.

OTHER METRICS

In addition to the previousely discussed metrics, there are others that deserve attention when evaluating soruce coude scanning and identfying

tools. They include advanced reporting capabilities, ability to incorporate company policy with the tool allowing easier flagging for what's allowed for ur or not based on integrated policy, ability to generate the applicable notices for the open source software bill of material, and the option to do modular installation and avoid the installation of various module (for instance security related or others) that are not of interest to you.

CONCLUSION

This chapter was created out of necessity and due to a lack of a unified way to evaluate source code scanning and license identifications tools. Table 11 (next page) provides a visual summary of these metrics. We hope that you find them helpful in capturing the important aspects of such tools when you are embarking on an evaluation journey of multiple tools and trying to decide which tool is more suitable for your needs and meets your requirements.

If you have suggestions for other metrics that should be covered in this chapter, please feel free to contact the author with your recommendation via **http://www.ibrahimatlinux.com/contact.html**.

Table 11. Summary of the key metrics used in evaluating source code scanning and license identification tools

Metric	Specifics
Knowledge Base	Size
	Frequency of update
Detection Capabilities	Whole components
	Partial snippets
	Ability to auto-identify code with proper origin and license
Ease of use	Intuitive, requires minimal amount of training
Operational Capabilities	Speed of scans
	Ability to use for M&A scans (no licensing lock on usage models)
	Support for different audit models
	Programming language agnostic

Metric	Specifics
Integration Capabilities	Integration with build systems via APIs and a CLI
	Integration of company compliance policies within the tool
Security Vulnerabilities Database	Size of database
	Frequency of update
	Sources of vulnerabilities information
	Research validating vulnerabilities' alerts
Advanced Discovery Methods	Support for advanced vulnerability discovery (i.e. identifying a vulnerability when vulnerable code was copied into a new component)
Cost	Infrastructure cost
	Operational cost
	Licensing cost
	Integration cost
	Lock-in cost
	Cost of engineering customization
Deployment models	On-site, Cloud, Hybrid
Other	Modular installation
	Generation of required notices
	Reporting capabilities

Chapter 13

OPEN SOURCE AUDITS IN MERGER AND ACQUISITION TRANSACTIONS

INTRODUCTION

We live in an era defined by software. Virtually everything we do on a daily basis is in some way planned, shaped, analyzed and managed by software. Within that large software umbrella, open source software is king. Companies across all industries are racing to use, participate in, and contribute to open source projects for the various advantages they offer, from the ability to leverage external engineering resources that accelerate time to market, to enabling faster innovation, and having capacity to focus on differentiating values.

The saying "Open Source is Eating the Software World" also applies to corporate transactions, as virtually any technology acquisition will involve software in some form. The software due diligence process, in which the acquirer performs a comprehensive review of the target's software and their compliance practices, is becoming a standard part of any merger or acquisition. During this process, it is common to come across open source software, which presents a set of verification challenges that are different from proprietary software. In this chapter, we provide an overview of the open source audit process in M&A transactions and offer recommendations on how to be better prepared for such a corporate transaction.

COMMON OPEN SOURCE USAGE SCENARIOS

Before diving into the open source due diligence process, it helps to understand the various ways developers incorporate open source software into the development process and build systems. This applies to situations where the company knowingly or unknowingly incorporates open source software into their source code base. The most common use scenarios of open source software are incorporation, linking, and modification.

Making changes to open source components, or injecting open source code in proprietary or third party components, can affect the way that audit service providers discover and report such code.

When engaging with an open source audit provider, it is often helpful to understand how their discovery approach captures open source code.

INCORPORATION

A developer may use a complete open source component or copy portions of a component into their software product's codebase. Since open source licenses come with a variety of obligations that may affect the company's legal responsibilities and the proprietary nature of their code, companies should track, declare and approve (internally) all such incorporation and use of open source code.

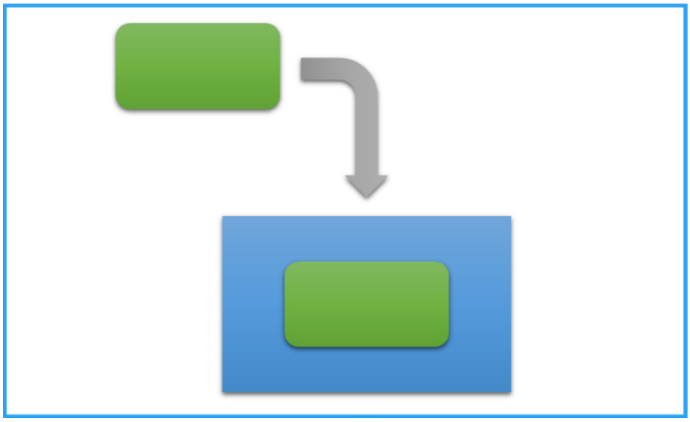


Figure 30. Incorporating open source code (green) within another body of code (blue)

The goal of a source code audit is to find all open source software incorporated into a software codebase, to avoid unpleasant surprises post-acquisition. The likelihood of undeclared incorporation increases when the target has not had sufficient developer training on open source compliance, or has relied upon transient worker like contractors or interns who do not maintain long-term records.

The incorporation scenario is often not obvious when human eyes look at source code, but source code scanning tools with the ability to discover and match snippets can easily uncover such incorporation.

LINKING

Linking is a very common scenario for instance when using open source libraries. In this scenario, a developer may link an open source software component with their software component.

Several terms can refer to such a scenario such as static/dynamic linking, combining, packaging, or creating interdependencies. It is often easy to detect linking when visually scanning source code because libraries are generally included at the beginnings of files and the linked code is likely to be in a separate named directory or file.

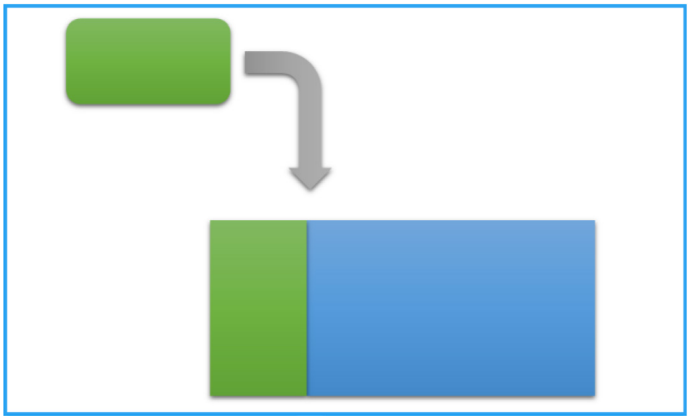


Figure 31. Linking open source code (green) within another body of code (blue)

Linking differs from incorporation in that the source code is kept separate, rather than being copied into a single combined form. Linking interactions happen either when the code is compiled into a single executable binary (static linking), or when the main program runs and calls the linked program (dynamic linking).

MODIFICATION

This is a very common scenario where a developer may make changes to an open source software component, including:

- Adding/injecting new code into the open source software component.
- Fixing, optimizing or making changes to the open source software component.
- Deleting or removing code.

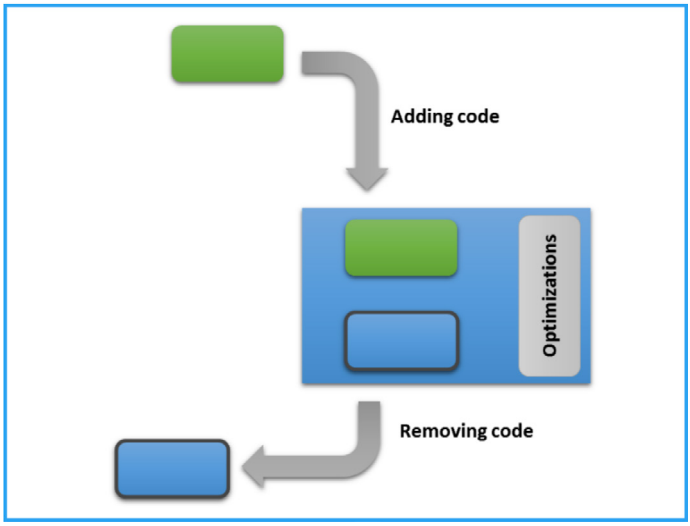


Figure 32. Modifications applied to open source code (green)

NOTE ON DEVELOPMENT TOOLS

It is important to be aware that certain development tools may perform some of these operations transparently. For example, a developer may use a development tool that automates certain portions of the development process. Examples of this include graphics frameworks that provide user interface templates, game development platforms that provide physics engines, or software development kits (SDKs) that provide connectors to cloud services. In order to provide these services, a tool must usually inject

portions of its own code into the developer's work product when the code is built. The license for such injected code by development tools should be verified especially given the resulting work is often statically linked.

OPEN SOURCE AUDITS

Every M&A transaction is different, but the need to verify the impact of acquiring open source obligations is a constant among all such transactions. Companies carry out open source audits to understand the depth of use and the reliance on open source software. In addition, such audits offer great insights about any compliance issues and even the engineering practices of the target company.

WHY CONDUCT AN OPEN SOURCE AUDIT?

Open source licenses may impose restrictions on how you can use and redistribute software. These may be incompatible with the acquiring company's business, and should be uncovered early. Examples of ways the presence of open source software can affect the acquired assets include:

- Open source licenses usually impose certain obligations that you must fulfill when you ship a product containing open source code.
- Some open source licenses require notices in documentation, or have restrictions for how you promote or advertise a product in relation to the open source code used in it.
- Failure to satisfy open source license obligations can lead to possible litigation, expensive re-engineering, product recalls, and bad publicity.

SHOULD YOU COMMISSION AN OPEN SOURCE AUDIT?

One common question is whether you need to commission an open source at all. The answer to that question differs by company, purpose of acquisition, and size of the source code. For instance, for small acquisitions, some companies (acquirers) prefer to just review the open source bill of materials (BoM) provided by the target (assuming it is available), and have

a discussion with their engineering lead about their open source practices. Even if the purpose of the acquisition is to acquire the talent, an audit can help uncover whether there are undisclosed liabilities due to historical license obligations from products that already shipped.

AUDIT INPUTS AND OUTPUTS

The audit process has one primary input and one primary output (Figure 33). The input to the process is the software stack subject to the M&A transaction. This includes proprietary, open source and third party software. On the end side of the process, the primary output is a detailed open source software bill of material that lists all open source software (components and snippets), their origin and confirmed licenses.

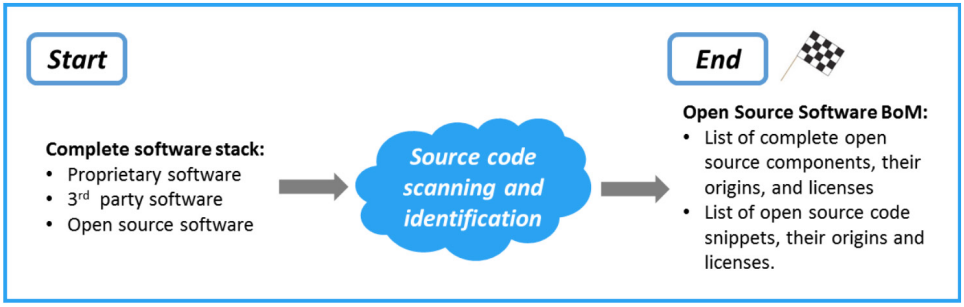


Figure 33. Inputs and outputs of the audit process

ASSESSING THE SCOPE OF AN AUDIT JOB

The size, scope, and cost of an audit varies by transaction, and generally increases with source code size and complexity. To provide a quote (cost and time) for an open source audit the auditor needs to get some basic understanding of the size and characteristics of the code base, as well as the urgency of the project.

The first questions from the auditor are specific to code metrics, such as the size of the source code base, the number of lines of source code, and the number of files that are included in the audit. They will also ask if the codebase consists exclusively of source code, or if it includes binary files, configuration files, documentation, and possibly other file formats.

Sometimes, it is also helpful for the auditor to know the file extensions subject to the audit.

Mature companies generally keep records about the open source components and versions used in their products and projects. Such information is very helpful and increases the understanding of the auditor on the expected workload.

Because audit price discussions happen early in the process based on size and scope, the acquirer may not have access to all the information described above.

At the very minimum, the auditor needs to understand the number of files to be scanned before proceeding, although additional information will help refine the estimates. When the auditor has enough information to understand the scope of the work, they will also need to understand the urgency, as this has a significant impact on the cost of an audit.

AUDIT METHODS

When performing an open source audit there are certain features in the tools that provide meaningful value to the acquirer. One of the most important features is the ability to search for open source code snippets that have been mixed into the proprietary code of the target company, and vice versa. Another feature is the ability to eliminate false positives from the audit results minimizing the amount of labor needed to so manually.

There are three audit methods:

- Traditional audit, in which the auditor gets complete access to all the code and executes the audit either remotely or on site.
- Blind audit, in which the auditor does the work remotely and without ever seeing the source code.
- “Do It Yourself” audit, where the target company or the acquirer performs most of the actual audit work themselves using the tools with the option for a random verification of the results performed by audit tool provider.

TRADITIONAL AUDIT METHOD

This method is the original method of source code scanning for open source compliance purposes and all audit service providers support it. Traditional audits are those where a compliance auditor from a 3rd party auditing company gets access to the source remotely via a cloud system or physically while visiting on site and performs the source code scan.

Figure 34 illustrates the audit process following the traditional auditing method. Please note that the process may vary slightly from one service provide to another.

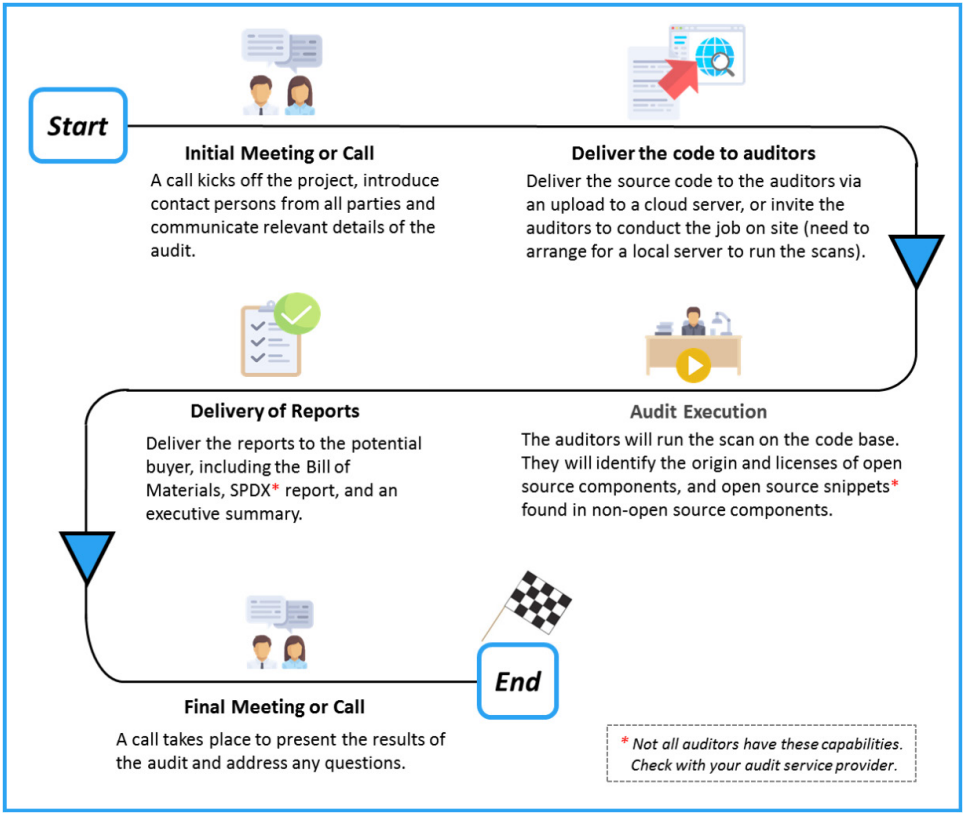


Figure 34. Illustration of generic audit process in the traditional audit method

A typical traditional audit process follows these steps:

- Auditor sends questions to the acquirer to have a better understanding of the job.
- Acquirer responds allowing auditor company to have a better understanding of the scope and audit parameters.
- Auditor provides quote based upon the responses.
- Agreement is reached on the quote. Next is singing service agreement, statement of work, non-disclosure agreement, etc.

Please note that "Start" in Figures 33, 34 and 35 assumes an actual start of the audit process when all partied involved have signed the contract and accompanying agreements.

- Auditor accesses to the target's code via secure cloud upload, or through a visit to the company for an on-site audit.
- Auditor scans the target's source code, cleans up the false positives, and evaluates the results.
- Auditor generates the report and delivers it to the client.
- A call or a face-to-face meeting follows to review the results with the auditor and address any questions.

Most of not all audit service providers support this audit method with varying degrees related to ability to discover snippets, support for programming languages, etc. This audit method allows the opportunity to collect multiple bids for the same audit job and the ability to choose the best bid given your requirements. Following this model, the target company must be willing to transfer the code to the auditors or allow them to visit your offices to complete the job on-site.

BLIND AUDIT

The blind audit method was pioneered by FOSSID, a Stockholm based company, to address the confidentiality requirements in M&A transactions.

Using their proprietary technology, FOSSID have the ability to perform audits and generate reports without having their engineers look at the source code subject to the audit.

Figure 35 illustrates the blind audit process designed to provide confidentiality of source code in M&A transactions. One major advantages of a blind audit include the ability for the auditor to complete the review without having access to the source code. In addition, with sufficient precautions by the acquirer, the auditor may also not gain awareness of the target's identity offering a high level of confidentiality. As far as the author is aware, outside of FOSSID, there are no other auditing service providers offering such or comparable service.

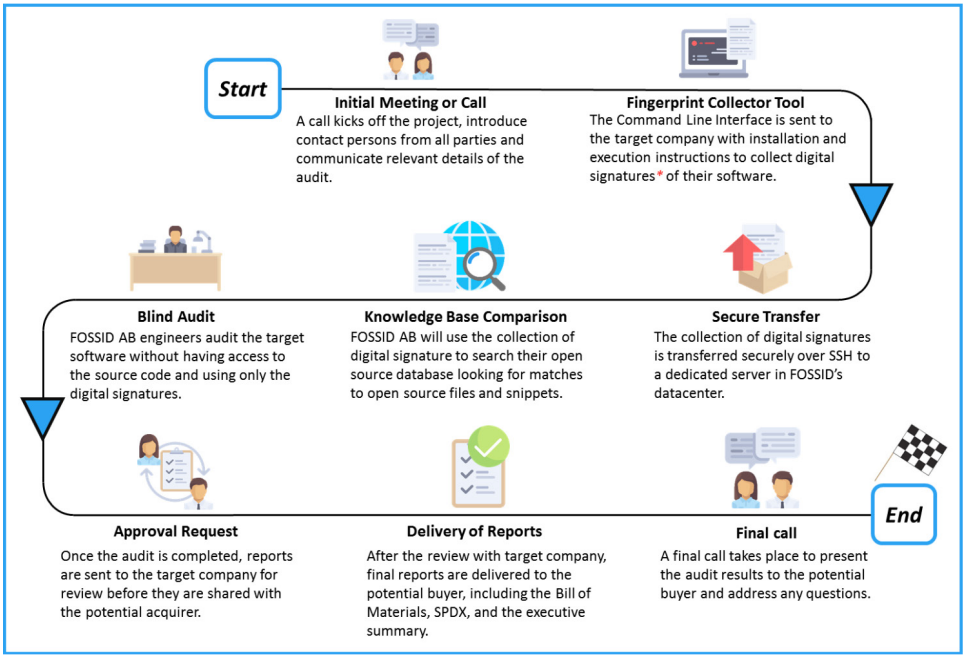


Figure 35. Illustration of a blind audit process

DIY AUDIT

The Do-It-Yourself (DIY) audit provides the acquirer or the target company time-limited access to the compliance cloud tools, enabling them to run the

scan themselves. They can then perform the audits internally with complete access to the knowledge base and all reporting facilities. This approach is particularly interesting for companies that have in-house employees with sufficient experience to interpret scan results and suggest remediation procedures. It can quickly become more cost-effective for companies that go through the M&A process several times per year. An independent certification can be performed to verify the findings, to further secure the integrity of the audit.

Figure 36 (next page) illustrates the DIY audit method using the tools from FOSSID. This approach has several advantages such as the ability to start the audit as soon as needed since it uses internal resources and not dependent on the availability of third party auditors, potentially shortening the timelines and reducing an external source of cost. Compliance concerns can be addressed immediately, since the people who have direct access to the code are performing the audit.

Finally, to verify the correctness and completeness the audit, the tool provider can perform a certain number of verification to that end. For instance, as part of their DIY offering, FOSSID offers the random verification of 1% of the files set forth to be audited by the target company.

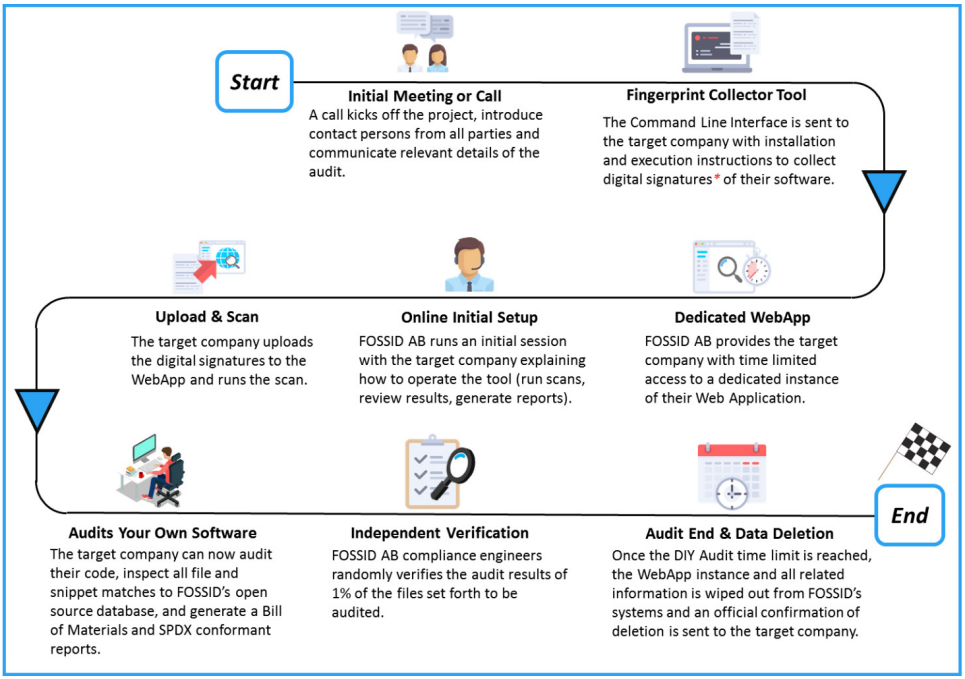


Figure 36. Illustration of a DIY audit process

NOTE ON THE FINAL REPORT

You can tune or configure many of the auditing tools can to flag or highlight potential scenarios that you can consider possible issues. After viewing the results carefully, you might find most of them to be non-issues. So be prepared for what might appear to be a lot of noise. The noise may come from things such as leftover code that is in the code tree but not used, sample code, or other. Therefore, the initial report may be lengthy and unfiltered and you should be prepared to invest time to filter the report to find the real issues.

As for SPDX, source code auditor often can provide an SPDX conformant report. Therefore, if you would like to receive one, you will need to request it early on, due to possible cost implications.

SECURITY AND VERSION CONTROL

It is a generally accepted truth that software ages like milk, not wine. Security vulnerabilities are a concern with all code whether it is open source or not. However, in open source projects these vulnerabilities are publicly exposed as well as the process of fixing them. Such an exposure can happen before or after the fix is implemented, and outdated open source code could potentially contain vulnerabilities that are actively exploited in the wild. While security and version control are not part of the open source compliance due diligence process, companies providing source code scanning services may also offer a service mapping identified open source components against known open source security vulnerabilities.

PRE- AND POST-ACQUISITION REMIDIATION

By this point, the acquiring company should have a clear idea how the target uses and manages open source software, and how successful they have been at satisfying their open source license obligations. The acquirer and target should use this information to negotiate remediation for any open source compliance issues. If any issues are uncovered in the audit, there are a few options for resolving them as a part of the pending transaction. The first option is to remove any offending code. If the open source software only augments proprietary code, it may be possible to eliminate it. Another option is to design around the offending component, or re-write any code using cleanroom techniques. If the section of code is truly essential or you have already previously distributed it, the only remaining option is to bring the code into compliance. The cost of such remediation efforts may be a factor to consider.

Whatever option you choose, it is crucial to identify the individuals who participated in incorporating the open source code, and to get them involved in the remediation effort. They might have additional documentation or knowledge that can be useful in resolving any issues.

PREPARING FOR AN AUDIT AS THE ACQUISITION TARGET

Passing an open source compliance audit is not hard if you are prepared. However, it is very unlikely to happen if you only begin preparing when an acquirer shows interest.

The open source compliance activities go hand-in-hand with your daily business and development activities. The objective of these activities is to ensure the company tracks all open source components, and respects open source license obligations resulting from your use of these open source components. These same measures can be of great help if your company becomes a target for a corporate transaction, as it minimizes the risk of surprises.

KNOW WHAT'S IN YOUR CODE

This is the golden rule of compliance. You need to maintain a complete software inventory for all software components including their origin and license information. This covers software components created by your organization, open source components, and components originating from third parties. The most important point is having a process for identifying and tracking open source components. You do not always need a complex compliance program; however, you should have five basic elements: policy, process, staff, training, and tools. We have discussed these elements earlier in this book.

BE IN COMPLIANCE

If you have shipped products containing open source software, whether intentionally or not, then you will need to comply with the various licenses governing those software components. Hence, the importance of knowing what is in your code, as a complete bill of materials makes compliance much easier.

Being in compliance is not a simple task, and it varies from product to product based upon the licenses and the structure of the code.

At a high level, being in compliance means that you:

- Track all use of open source software.
- Compile a finalized open source BoM for all software in the shipping image of product.
- Fulfill the obligations of the open source licenses.
- Repeat the process every time you issue a software update.
- Respond quickly and seriously to compliance inquiries.

USE LATEST SOFTWARE RELEASES

One of the benefits of a comprehensive compliance program is that it is easier to find products with insecure versions of open source components and replace them. Most source code scanning tools now provide functionality to flag security vulnerabilities disclosed in older software components. One important consideration when upgrading an open source component is to ensure that the component retains the same license as the previous version. Open source projects have occasionally changed licenses on major releases. To avoid a situation where you are using a version with security vulnerabilities, companies are encouraged to engage with open source project communities. It is not reasonable or feasible to be active in all of the open source projects you use; therefore, a certain level of prioritization is needed to identify the most critical components. There are various levels of engagement, ranging from joining mailing lists and participating in the technical discussions, to contributing bug fixes and small features, to major contributions. At minimum, it is very beneficial for corporate developers working on a specific open source project to subscribe to and monitor the mailing list for any reports related to security vulnerabilities, and available fixes.

MEASURE UP YOUR COMPLIANCE EFFORTS

The easiest and most effective first step for organizations of all sizes is to engage with the OpenChain Project and to obtain “OpenChain Conformant” (**https://www.openchainproject.org/conformance**) status. Companies can do that by filling out a series of questions either online (**https://certification.openchainproject.org/**) or manually (**https://wiki.**

linuxfoundation.org/_media/openchain/openchain_conformance_conformance_check_1.1.pdf). The questions used for OpenChain Conformance help to confirm that an organization has created processes or policies for open source software compliance.

OpenChain is an industry standard, similar to ISO 9001. It focuses on the "big picture," with precise processes and policy implementations up to each individual organization. OpenChain Conformance shows that open source compliance processes or policies exist, and that further details can be shared when requested by a supplier or customer. OpenChain is designed to build trust between organizations across the global supply chain.

The Linux Foundation's Self-Assessment Checklist is an extensive checklist of compliance best practices, in addition to elements that must be available in an open source compliance program to ensure its success. Companies can use this resource as an internal, self-administered checklist to evaluate their compliance in comparison to compliance best practices.

PREPARING FOR AN AUDIT AS THE ACQUIRING COMPANY

As an acquirer, there are actions to take and decisions to make before the audit is commissioned, and then after you receive the results.

CHOOSE THE RIGHT AUDIT MODEL AND AUDITORS FOR YOUR NEEDS

As previously discussed, there are three primary audit methods that can be used and you will need to decide which is most suited to your specific situation, given the parameters you are working with.

PRIORITIZE WHAT YOU CARE ABOUT

The audit report may provide a significant amount of information, depending on the complexity of the scanned code. It is important to identify which licenses and use-cases are regarded as critical.

ASK THE RIGHT QUESTIONS

The open source audit report offers a lot of information about the target's source code and the licenses involved. However, there are many other data points that will require further investigation to get clarifications or confirmations on compliance related concerns. In this section, we offer a collection of questions as a starting point to frame what is important to you, and what questions you should address with the target company.

Some of the questions you need to address as part of this process include:

- Has the target used code with licenses that could jeopardize the IP of the target or acquirer?
- Are there any snippets with unknown origin and/or unknown license?
- Are the target's open source compliance practices sufficiently mature and comprehensive?
- Does the target company track known vulnerabilities in their open source components?
- When distributing products, does the target provide all necessary materials to satisfy open source license obligations (written offer, various required notices, and source code when applicable)?
- Does the target company's compliance process aligned with the speed of development to meet product release schedules?
- Does the target have a process in place to respond to all internal and external requests for source code in a timely manner?

IDENTIFY AND SOLVE CONCERNS

In some cases, an open source audit may reveal instances of licenses or compliance practices that are not acceptable to the acquirer. The acquirer can then request these instances to be mitigated as a condition for closing. For instance, the target company may use a code component that comes license under "License A", but the acquiring company has a strict policy

against using any source code licensed under “License A”. In such a situation, both parties will need to discuss and figure out a possible solution.

POST-ACQUISITION IMPROVEMENT PLAN

This is especially important when the acquirer is a large company buying a smaller startup that will continue to operate as a subsidiary. In this scenario, the acquirer often helps the target establish a formal compliance policy and process, provides training on their own practices, and offers ongoing guidance and support.

CONCLUSION

Open source due diligence is generally one task in a long list of tasks that need to be successfully completed in an M&A transaction. However, it is still an important aspect of the general due diligence exercise given the central role of software and potential IP risks. While the open source due diligence may seem a lengthy process, it often can be completed quickly, especially if both parties are prepared, and working with a swift compliance service provider.

How can you be prepared?

If you are the target, you can maintain proper open source compliance practices by ensuring your development and business processes include:

- Identifying the origin and license of all internal and external software.
- Tracking open source software within the development process.
- Performing source code reviews for new or updated source code entering the build.
- Fulfilling license obligations for open source components when a product ships or when software is updated.
- Offering open source compliance training to employees.

If you are the acquirer, you should know what to look for and have the skills on-hand to address issues quickly:

- Decide with the target company on the appropriate audit method to use, and which third party service provider to engage for the audit. Note that some compliance auditing service providers do not have ability to do blind testing, some may support the DIY, and others do not have the ability to discover code snippets.
- If possible, you should aim for multiple bids for the project. It is not just about the cost, but also about having the precise output that will help you address any concerns you may have.
- Make sure you have the internal expertise to compare bids equally and fairly, and that they include all audit parameters such as:
    - Audit method, inputs and outputs
    - Primary contact persons at target and acquirer for speedy discussions of issues that arise
    - Timeline and logistics especially if it involves an on-site visit
    - Confidentiality parameters
    - Code vulnerabilities and version control analysis
    - Cost, normal process and expedited

Open source compliance is an ongoing process, not a destination. Maintaining good open source compliance practices enables companies to be ready for any scenario where software changes hands, from a possible acquisition, a sale, or product or service release. For this reason and many others, companies are highly encouraged to invest in building and improving upon their open source compliance programs.

REFERENCES

The Linux Foundation Open Compliance Program
https://compliance.linuxfoundation.org/

Free E-book: Practical GPL Compliance
Published by The Linux Foundation, Practical GPL Compliance is a compliance guide for startups, small businesses, and engineers, particularly focused on complying with the versions of the GNU General Public License (GPL). Its goal is to provide practical information and quickly address common issues.
https://www.linuxfoundation.org/news-media/research/practical-gpl-compliance

OpenChain
OpenChain identifies common best practices in open source compliance that should be applied as a standard across a supply chain
https://openchainproject.org

OpenChain Curriculum
The OpenChain Curriculum help organizations meet the training and process requirements of the OpenChain Specification. It is also a general open source training and – because of its public domain licensing – you can re-use it for internal or external purposes without any restrictions.
https://wiki.linuxfoundation.org/openchain/curriculum

Free Training: Compliance Basics for Developers
A free open source compliance course from the Linux Foundation targeted for developers.
https://training.linuxfoundation.org/linux-courses/open-source-compliance-courses/compliance-basics-for-developers

Software Package Data Exchange® (SPDX)
SPDX is a set of standard format for communicating the components, licenses and copyrights of software packages.
https://spdx.org/

Self-Assessment Checklist
The Linux Foundation has compiled this extensive checklist of compliance practices found in industry-leading compliance programs. Companies can use this checklist as a confidential internal tool to assess their progress in implementing a rigorous compliance process and to help them prioritize process-improvement efforts.
https://go.linuxfoundation.org/self-assessment-checklist

TODO Group

TODO is an open group of companies that collaborate on practices, tools, and other ways to run successful and effective open source programs.

http://todogroup.org/

Using Open Source

This enterprise guide by The Linux Foundation offers practical guidance on using open source software in a legal and responsible way.

https://www.linuxfoundation.org/resources/open-source-guides/using-open-source-code/

A Template for Approval Request Form For The Use of Free and Open Source Software

This document is part of the free resources made available by The Linux Foundation Open Compliance Program. It offers a template for the Approval Request Form used by developers to request approval to use Free and Open Source Software (FOSS) in a commercial product. The company's Open Source Review Board (OSRB) reviews the submission and determines approval. In most cases, the submission, review and approval of such requests is managed via an online system that is part of the company's FOSS compliance management process.

https://www.linuxfoundation.org/events/a-template-for-approval-request-form-for-the-use-of-free-and-open-source-software/

Generic FOSS Policy

Companies using FOSS often create a company-wide policy to ensure that all staff is informed of how to use FOSS (especially in products), to maximize the impact and benefit of using FOSS, and to ensure that any technical, legal, or business risks resulting from that usage are properly mitigated. This document is a free resource available from the Linux Foundation under the Open Compliance Program. It offers a generic FOSS policy that companies can use as starting point in creating their own FOSS policy. It provides a template policy that focuses on governing FOSS usage in externally distributed products and that can be customized to the company's specific needs.

https://wiki.linuxfoundation.org/_media/openchain/lf_compliance_generic_foss_policy.pdf

ABOUT THE AUTHOR

Ibrahim Haddad (Ph.D.) is Vice President of Strategic Programs at the Linux Foundation. In this role, he works with the largest technology companies and open source communities to facilitate a vendor-neutral environment for advancing the Linux and open source platform.

Twitter: **@IbrahimAtLinux**
Web: **IbrahimAtLinux.com**

CONTRIBUTORS

Shane Coughlan is an expert in communication, security and business development. His experience includes engagement with the enterprise, embedded, mobile and automotive industries. Shane has extensive knowledge of open source governance, internal process development, supply chain management and community building. He currently leads the OpenChain Project community.

Twitter: **@opendawn**

Kate Stewart works at the Linux Foundation on Strategic Programs. She has over 30 years of experience working as a developer, release manager, and director of product management in the embedded ecosystem before joining the Linux Foundation. Managing open source software development teams in the US, Canada, UK, India, and China has taught her that we need to be pragmatic about open source compliance, and get it automated, which was why she worked with others to start SPDX.

Twitter: **@_kate_stewart**