

** Yellow marker : Difference between Release2(for Spec1.0)



OPENCHAIN
CURRICULUM

FOSS Training Reference Slides for the OpenChain Specification 1.1

Released under CC0-1.0.

You may use, modify, and share these slides without restriction.
They also come with no warranty.

These slides follow US law. Different legal jurisdictions may have different legal requirements.
This should be taken into account when using these slides as part of a compliance training program.

These slides do not contain legal advice

Welcome to the OpenChain Curriculum Slides. These slides can be used to help train internal teams about FOSS compliance issues and to conform with the OpenChain Specification.

You can deliver these slides as one half-day training session or you can deliver each chapter as a separate module. Please note that each chapter has “Check Your Understanding” slides with questions and answers in the slide notes. These can be used as the basis for in-house tests for FOSS compliance.



What is the OpenChain Curriculum?

- The OpenChain Project helps to identify and share the core components of a Free and Open Source Software (FOSS) compliance program.
- The core of the OpenChain Project is the Specification. This identifies and publishes the core requirements a FOSS compliance program should satisfy.
- The OpenChain Curriculum supports the Specification by providing freely available training material.
- These slides help companies satisfy the requirements of the Specification Section 1.2. They can also be used for general compliance training.

Learn more at: <https://www.openchainproject.org>

This slide helps explain what the OpenChain Curriculum and these slides are for.

Contents

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

This slide is relevant to providing either a single three hour training session or explaining how a series of shorter sessions focused on “per chapter” training will work.

FOSS Policy

- <<This is a placeholder slide to identify where your FOSS policy can be found (OpenChain Specification 1.1, section 1.1.1)>>
- You can get an example FOSS policy via the Linux Foundation Open Compliance Program at:
<https://www.linux.com/publications/generic-foss-policy>

This slide is intended to help a company identify where their internal FOSS policy is located in the company documentation.

**Note was deleted.

CHAPTER 1

What is Intellectual Property?

What is “Intellectual Property”?

- Copyright: protects original works of authorship
 - Protects expression (not the underlying idea)
 - It covers software, books, and similar works
- Patents: useful inventions that are novel and non-obvious
 - Limited monopoly to incentivize innovation
- Trade secrets: protects valuable confidential information
- Trademarks: protects marks (word, logos, slogans, color, etc.) that identify the source of the product
 - Consumer and brand protection; avoid consumer confusion and brand dilution

*This chapter will focus on copyright and patents,
the areas most relevant to FOSS compliance.*

This overview is not intended to cover all aspects of Intellectual Property. It is intended to provide context for the “big picture” and to establish that today we are only discussing copyright and patents, the areas most relevant to FOSS compliance.

Copyright Concepts in Software



- Basic rule: copyright protects creative works 
 - Copyright generally applies to literary works, such as books, movies, pictures, music, maps
 - Software is protected by copyright
-  Not the functionality (that's protected by patents) but the expression (creativity in implementation details)
 - Includes Binary Code and Source Code
- The copyright owner only has control over the work that he or she created, not someone else's independent creation
 - Infringement may occur if copying without the permission of the author

This slide explains the “big picture” of copyright in software.

Copyright Rights Most Relevant to Software



- The right to *reproduce* the software – making copies
- The right to create "*derivative works*" – making modifications
 - The term derivative work comes from the US Copyright Act
 - It is a "term of art" meaning that it has a particular meaning based on the statute and not the dictionary definition
 - In general it refers to a new work based upon an original work to which enough original creative work has been added so that the new work represents an original work of authorship rather than a copy
- The right to *distribute*
 - Distribution is generally viewed as the provision of a copy of a piece of software, in binary or source code form, to another entity (an individual or organization outside your company or organization)

Note: The interpretation of what constitutes a "derivative work" or a "distribution" is subject to debate in the FOSS community and within FOSS legal circles

This slide clarifies the most important parts of copyright law to software.

Patent Concepts in Software



- Patents protect functionality – this can include a method of operation, such as a computer program
 - Does not protect abstract ideas, laws of nature
- A patent application must be made in a specific jurisdiction in order to obtain a patent in that country. If a patent is awarded, the owner has the right to stop anybody from exercising its functionality, regardless of independent creation
- Other parties who want to use the technology may seek a patent license (which may grant rights to use, make, have made, sell, offer for sale, and import the technology)
- Infringement may occur even if other parties independently create the same invention

This slide explains patent concepts relevant to software.

Licenses

- A “license” is the way a copyright or patent holder gives permission or rights to someone else
- The license can be limited to:
 - Types of use allowed (**commercial / non-commercial**, distribution, derivative works / to make, have made, manufacture)
 - Exclusive or non-exclusive terms
 - Geographical scope
 - Perpetual or time limited duration
- The license can have conditions on the grants, meaning you only get the license if you comply with certain obligations
 - E.g, provide attribution, or give a reciprocal license
- May also include contractual terms regarding warranties, indemnification, support, upgrade, maintenance

This slide explains what is a “license.” This is different to a contract under US law. This slide explains the boundaries of what can be in a license.

Check Your Understanding

- 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 might need a copyright license from a third party for that software?
A patent license?

Copyright protects original works of authorship. It's different than patent in that copyright protects the expression of an idea, whereas patent protects the underlying idea itself. Examples of works of authorship include photographs, songs, and computer code.

Most important copyright concepts for software are: right to reproduce, right to make creative works (or right to modify), and right to distribute.

Software can be subject to a patent. Patent protects method of operation, such as computer program. However, patent protects functionality, and not abstract ideas.

Patent holder can exclude others from practicing the patent, regardless of whether the others have independently created the product.

If you have independently developed your own software, then you may not need a copyright license if you can show the independent development and you had no access to the copyrighted work in question. This is difficult if the copyrighted work is popular such that it'd be reasonable to assume that you

had access. If your software reads on a patent, then you will need a patent license regardless of whether you've independently developed the software. An example of this would be FFmpeg, which is a free software project that provides the codecs for encoding and decoding videos. However, you would still need a patent license to encode and decode a certain format.

** Note was deleted.

CHAPTER 2

Introduction to FOSS Licenses

FOSS Licenses

- FOSS licenses **by definition** make source code available under terms that allow for modification and redistribution
- FOSS licenses may have conditions related to providing attributions, copyright statement preservation, or a written offer to make the source code available
- One popular set of licenses are those approved by the FOSS Initiative (OSI) based on their FOSS Definition (OSD). A complete list of OSI-approved licenses is available at <http://www.opensource.org/licenses/>

This slide provides the “big picture” about what FOSS licenses do. It also explains a resource where you can find out more about some FOSS licenses.

Permissive FOSS Licenses

- Permissive FOSS license: a term used often to describe minimally restrictive FOSS licenses
- Example: BSD-3-Clause
 - The BSD license is an example of a permissive license that allows unlimited redistribution for any purpose **in source or object code form** as long as its copyright notices and the license's disclaimers of warranty are maintained
 - The license contains a clause restricting use of the names of contributors for endorsement of a derived work without specific permission
- Other examples: MIT, Apache-2.0

This slide explains "permissive" FOSS licenses, the most basic type of FOSS license, which usually have minimal requirements. The most basic requirement is to include a copyright notice. Permissive licenses do not require source code to be made available to downstream recipients. The code owner is providing the source code under the FOSS license, but is not requiring that you provide the source code to others.

License Reciprocity & Copyleft Licenses

- Some licenses require **that if** derivative works (or software in the same file, same program or other boundary) **are distributed, the distribution is** under the same terms as the original work
- This is referred to as a “**copyleft**” or “reciprocal” effect 
- Example of license reciprocity from the GPL version 2.0:
You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed [...] under the terms of this License.
- Licenses that include reciprocity or Copyleft clauses include all versions of the GPL, LGPL, AGPL, MPL and CDDL



This slide explains reciprocity and Copyleft, a more complex type of FOSS license that have additional requirements above permissive licenses. They require distribution of the original work and derivative works under the same terms as the original work.

Proprietary License or Closed Source

- A proprietary software license (or commercial license or EULA) has restrictions on the usage, modification **and/or** distribution of the software
- Proprietary licenses are unique to each vendor – there are as many variations of proprietary licenses as there are vendors and each must be evaluated individually
- FOSS developers often use the term “proprietary” to describe a commercial non-FOSS license, even though both FOSS and proprietary licenses are based on intellectual property and provide a license grant to that property



This slide explains proprietary or closed source licenses. These licenses often have very different requirements and rules compared to FOSS licenses.

Other Non-FOSS Licensing Situations

- Freeware – software distributed under a proprietary license at no or very low cost
 - The source code may or may not be available, and creation of derivative works is usually restricted
 - Freeware software is usually fully functional (no locked features) and available for unlimited use (no locking on days of usage)
 - Freeware software licenses usually impose restrictions in relation to copying, distributing, and making derivative works of the software, as well as restrictions on the type of usage (personal, commercial, academic, etc.)
- Shareware – proprietary software provided to users on a trial basis, for a limited time, free of charge and with limited functionalities or features
 - The goal of shareware is to give potential buyers the opportunity to use the program and judge its usefulness before purchasing a license for the full version of the software
 - Most companies are very leery of Shareware, because Shareware vendors often approach companies for large license payments after the software has freely propagated within their organizations.



There are other types of license used. Sometimes these are confused with FOSS but their requirements are actually different. Freeware or Shareware licensing should not be regarded as the same or compatible with FOSS licensing.

** This page was newly added.(Note is the same as previous page)

Other Non-FOSS Licensing Situations



- “Non-commercial” – some licenses have most of the characteristics of a FOSS license, but are limited to non-commercial use (e.g. CC-BY-NC).
 - FOSS by definition cannot limit the field of use of the software
 - Commercial use is a field of use so any restriction prevents the license from being FOSS

There are other types of license used. Sometimes these are confused with FOSS but their requirements are actually different. Freeware or Shareware licensing should not be regarded as the same or compatible with FOSS licensing.

Public Domain

- The term **public domain** refers to **software** not protected by law and therefore usable by the public without requiring a license
- Developers may include a *public domain declaration* with their software
 - E. g., "All of the code and documentation in this software has been dedicated to the public domain by the authors."
 - The public domain declaration is not the same as a FOSS license
- A public domain declaration attempts to waive or eliminate any intellectual property rights that the developers may have in the software to make it clear that it can be used without restriction, but the enforceability of these declarations is subject to dispute within the FOSS community and its effectiveness at law varies from jurisdiction to jurisdiction
- Often the public domain declaration is accompanied by other terms, such as warranty disclaimers; in such cases, the software may be viewed as being under a license rather than being in the public domain

This slide explains public domain, a type of release that means the work is released without any restrictions whatsoever by the authors. In the US public domain software can be included in FOSS code, but it should be noted that not all legal jurisdictions recognize the existence or permit the release of authorship under public domain. Germany is one example.

License Compatibility

- License compatibility is the process of ensuring that license terms do not conflict.
- If one license requires you to do something and another prohibits doing that, the licenses conflict and are not compatible if the combination of the two software modules trigger the obligations under a license.
 - GPL-2.0 and EPL-1.0 each extend their obligations to “derivative works” which are distributed.
 - If a GPL-2.0 module is combined with an EPL-1.0 module and the merged module is distributed, that module must
 - (according to GPL-2.0) be distributed under GPL-2.0 only, and
 - (according to EPL-1.0) under EPL-1.0 only.
 - The distributor cannot satisfy both conditions at once so the module may not be distributed.
 - This is an example of *license incompatibility*.

The definition of “derivative work” is subject to different views in the FOSS community and its interpretation in law is likely to vary from jurisdiction to jurisdiction.

This slide explains license compatibility, the way of understanding what licenses can be used together. Some FOSS licenses are compatible with each other. Some are incompatible. This is an important consideration when choosing code and choosing licenses.

Notices

Notices, such as text in comments in file headers, often provide authorship and licensing information. FOSS licenses may also require the placement of notices in or alongside source code or documentation to give credit to the author (an attribution) or to make it clear the software includes modifications.

- **Copyright notice** – an identifier placed on copies of the work to inform the world of copyright ownership. Example: Copyright © A. Person (2016)
- **License notice** – a notice that specifies and acknowledges the license terms and conditions of the FOSS included in the product.
- **Attribution notice** – a notice included in the product release that acknowledges the identity of the original authors and / or sponsors of the FOSS included in the product.
- **Modification notice** – a notice that you have made modifications to the source code of a file, such as adding your copyright notice to the top of the file.

This slide explains notices, the text comments in files that explain authorship and licensing, and which are often regarded as the most important way of knowing what license applies to a file.

Multi-Licensing

- Multi-licensing refers to the practice of distributing software under two or more different sets of terms and conditions **simultaneously**
 - E.g., when software is “dual licensed,” **the copyright owner gives each recipient** the choice of two licenses
- Note: This should not be confused for situations in which a licensor imposes more than one license, and you must comply with **all** of them



This slide explains multi-licensing. This is the situation where more than one set of license terms can apply to a piece of software.

Conjunctive = Multiple licenses apply

GPL-2.0 project also includes code under BSD-3-Clause

In this situation you have to comply with both sets of license terms

Disjunctive = Choice of one open source license or another

Mozilla tri-license

Jetty

Ruby

Disjunctive licensing may be something important to explore more deeply when creating a FOSS policy.

Under disjunctive licensing you have a choice of licensing, i.e. GPL and a more permissive license option, you may choose which license you are going to distribute under depending on license compatibility, license requirements. Sometimes a project has a disjunctive licensing situation, but only one license is included in your code – so perhaps the person you got the code from

already made this choice. If they choose the license you weren't going to use, now you might have to consider if you should figure out who the original © holder is and get the code directly from them

Example:

MPL 1.1/GPL 2.0/LGPL 2.1 --

"The contents of this file are subject to the Mozilla Public License Version - 1.1 (the "License"); you may not use this file except in compliance with - the License.

...

Alternatively, the contents of this file may be used under the terms of - either the GNU General Public License Version 2 or later (the "GPL"), or - the GNU Lesser General Public License Version 2.1 or later (the "LGPL"), - in which case the provisions of the GPL or the LGPL are applicable instead - of those above.

If you wish to allow use of your version of this file only - under the terms of either the GPL or the LGPL, and not to allow others to - use your version of this file under the terms of the MPL, indicate your - decision by deleting the provisions above and replace them with the notice - and other provisions required by the LGPL or the GPL. If you do not delete - the provisions above, a recipient may use your version of this file under - the terms of any one of the MPL, the GPL or the LGPL. "

"dual" = confusing term that may be used for any of these situations, but usually refers to business model of OSS license or commercial license choice
For more on dual-licensing as a business model: <http://oss-watch.ac.uk/resources/duallicenc2>

Check Your Understanding

- What is a FOSS license?
- What are typical obligations of a permissive FOSS license?
- Name some permissive FOSS licenses.
- What does license reciprocity mean?
- Name some copyleft-style licenses.
- What needs to be distributed for code used under a copyleft license?
- Are Freeware and Shareware software considered FOSS?
- What is a multi-license?
- What information may you find in FOSS Notices, and how may the notices be used?

FOSS licenses are Free and FOSS Software licenses generally make source code available under terms that allow for modification and redistribution.

Typical obligations of a permissive FOSS license are that the copyright notice and warranty disclaimer are included with the software. Very often, the license would expressly prohibits users from using the author's name without permission.

Examples of permissive FOSS licenses include MIT, BSD, and Apache.

License reciprocity means that the derivative work of the copyrighted work must be made available under the same license. Other names being used include "hereditary", "copyleft", "share-alike", and pejoratively "viral."

Examples of copyleft-style licenses include GPL and LGPL.

Copyleft-style licenses often have source availability obligations, which require you to provide accompanying source code when you distribute a binary version of a program or library. The source code should be of the same

version and content that corresponds to the binary version you distribute.

Freeware and Shareware are not FOSS. The reason is that even though freeware and shareware are available without cost, they don't allow the users to make modifications to the software. In fact, many of the freeware and shareware contain similar license restrictions common in proprietary software.

Multi-license refers to the practice where software is made available under multiple licenses. For example, an open source software can be dual-licensed under MIT and GPLv2. In that case, you are free to choose the license that suits your need.

FOSS Notices may include information about the identity of the copyright holders and the license governing the software. FOSS Notices may provide notice about modifications. Some licenses require that FOSS Notices be retained or reproduced for attribution purposes.

**Note was deleted.

CHAPTER 3

Introduction to FOSS Compliance

FOSS Compliance Goals

- Know your obligations. You should have a process for identifying and tracking FOSS components that are present in your software
- Satisfy license obligations. Your process should be capable of handling FOSS license obligations that arise from your organization's business practices

This slide explains that FOSS compliance is really a two-part goal. The first is to know your obligations and have a process to support this knowledge. The second is to satisfy the obligations.

What Compliance Obligations Must Be Satisfied?



Depending on the FOSS license(s) involved, your compliance obligations may consist of:

- **Attribution and Notices.** You may need to provide or retain copyright and license text in the source code and/or product documentation or user interface, so that downstream users know the origin of the software and their rights under the licenses. You may also need to provide notices regarding modifications, or full copies of the license.
- **Source code availability.** You may need to provide source code for the FOSS software, for modifications you make, for combined or linked software, and scripts that control the build process.
- **Reciprocity.** You may need to maintain modified versions or derivative works under the same license that governs the FOSS component.
- **Other terms.** The FOSS license may restrict use of the copyright holder name or trademark, may require modified versions to use a different name to avoid confusion, or may terminate upon any breach.

This slide expands on what compliance obligations must be satisfied in typical FOSS licenses.

The scope of source code availability is determined by the FOSS license. Some licenses may require source code availability for only the FOSS software. Others may require all the software described in the slide.



FOSS Compliance Issues: Distribution

- Dissemination of material to an outside entity
 - Applications downloaded to a user's machine or mobile device
 - JavaScript, web client, or other code that is downloaded to the user's machine
- For some FOSS licenses, access via a computer network can be a "trigger" event
 - Some licenses define the trigger event to include permitting access to software running on a server (e.g., all versions of the Affero GPL if the software is modified) or in the case of "users interacting with it remotely through a computer network"

This slide explains when FOSS obligations are “triggered.” FOSS licenses are copyright licenses and the basic compliance trigger is when you distribute code to another legal entity.

FOSS Compliance Issues: Modification

- Changes to the existing program (e.g., additions, deletions of code in a file, combining components together)
- Under some FOSS licenses, modifications may cause additional obligations upon distribution, such as:
 - Providing notice of modification
 - Providing accompanying source code
 - Licensing modifications under the same license that governs the FOSS component

This slide explains that modifying code can impose obligations under FOSS licenses. It explains a little bit about derivative works.

FOSS Compliance Program

Organizations that have been successful at FOSS compliance have created their own *FOSS Compliance Programs* (consisting of policies, processes, training and tools) to:

1. Facilitate effective usage of FOSS in **their products (commercial or otherwise)**
2. Respect FOSS developer/owner rights and comply with license obligations
3. Contribute **to** and participate in **FOSS** communities

This slide explains how FOSS compliance programs work in “broad strokes” (a basic overview).

Implementing Compliance Practices

Prepare business processes and sufficient staff to handle:

- Identification of the origin and license of all internal and external software
- Tracking FOSS software within the development process
- Performing FOSS review and identifying license obligations
- Fulfillment of license obligations when product ships
- Oversight for FOSS Compliance Program, creation of policy, and compliance decisions
- Training

This slide explains more about how FOSS compliance practices can work in an organization.

Compliance Benefits

Benefits of a robust FOSS Compliance program include:

- Increased understanding of the benefits of FOSS and how it impacts your organization
- Increased understanding of the costs and risks associated with using FOSS
- Increased knowledge of available FOSS solutions
- Reduction and management of infringement risk, increased respect of FOSS developers/owners' licensing choices
- Fostering relationships with the FOSS community and FOSS organizations

This slide describes some of the benefits that compliance brings to an organization beyond the fact of fulfilling the legal obligations of the license.

Check Your Understanding

- What does FOSS compliance mean?
- What are two main goals of a FOSS Compliance Program?
- List and describe important business practices of a FOSS Compliance Program.
- What are some benefits of a FOSS Compliance Program?

FOSS compliance means following the licensing terms of FOSS licenses. It involves understanding the licenses, having processes to support the license terms, and having processes to address any oversights or errors.

The two main goals of a FOSS compliance program are **know your obligations** and to **satisfy your obligations**.

The important business practices of a FOSS compliance program include:

- Identification of the origin and license of FOSS software
- Tracking FOSS software within the development process
- Performing FOSS review and identifying license obligations
- Fulfillment of license obligations when product ships
- Oversight for FOSS Compliance Program, creation of policy, and compliance decisions
- Training

A FOSS compliance program provides various benefits such as an increased understanding of how FOSS impacts your organization, an increased understanding of the costs and risks associated with FOSS, better relations

with the FOSS community and increased knowledge of available FOSS solutions.

**Note was deleted.

CHAPTER 4

Key Software Concepts for FOSS Review

How do you want to use a FOSS component?

Common scenarios include:

- Incorporation
- Linking
- Modification
- Translation

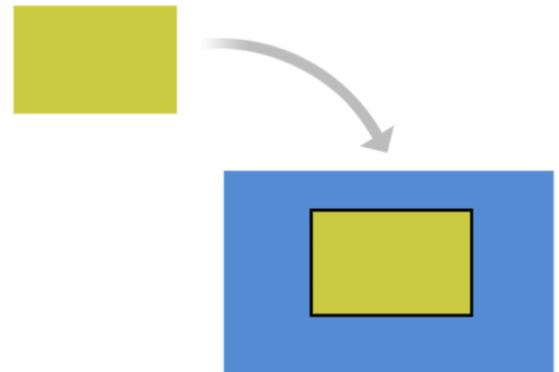
This slide is about how the use of FOSS components is a consideration for your compliance. Different use cases will have different legal effects. The next few slides explain these concepts in more detail.

Incorporation

A developer may copy portions of a FOSS component into your software product.

Relevant terms include:

- Integrating
- Merging
- Pasting
- Adapting
- Inserting



This slide outlines what incorporation means when using FOSS.

Linking

A developer may link or join a FOSS component with your software product.

Relevant terms include:

- Static/Dynamic Linking
- Pairing
- Combining
- Utilizing
- Packaging
- Creating interdependency

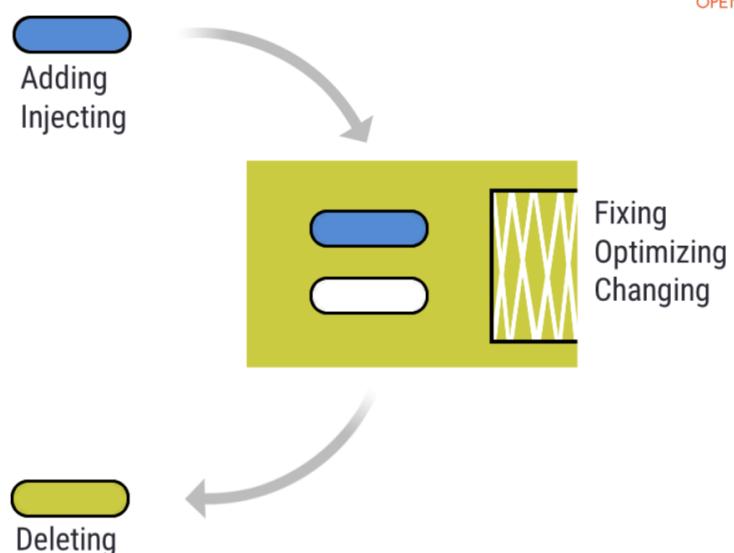


This slide outlines what linking means when using FOSS.

Modification

A developer may make changes to a FOSS component, including:

- Adding/injecting new code into the FOSS component
- Fixing, optimizing or making changes to the FOSS component
- Deleting or removing code



This slide outlines what modification means when using FOSS.

Translation

A developer may transform the code from one state to another.

Examples include:

- Translating Chinese to English
- Converting C++ to Java
- Compiling into binary

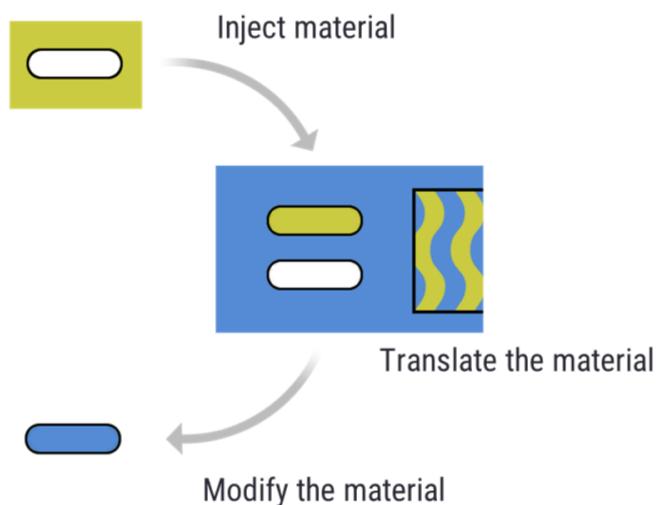


This slide outlines what translation means when using FOSS.

Development Tools

Development tools may perform some of these operations behind the scenes.

For example, a tool may inject portions of its own code into output of the tool.



This slide explains that development tools may do some of these actions “behind the scene”, and this is an area that companies should be aware of.

How is the FOSS component distributed?

- Who receives the software?
 - Customer/Partner
 - Community project
 - Another legal entity within the business group (this may count as distribution)
- What format for delivery?
 - Source code delivery
 - Binary delivery
 - Pre-loaded onto hardware

This slide explains some of the concepts behind distribution. Because FOSS licenses usually apply during distribution, this is a key point to consider in a compliance program.

Check Your Understanding

- What is incorporation?
- What is linking?
- What is modification?
- What is translation?
- What factors are important in assessing a distribution?

Incorporation is when you copy portions of a FOSS component into your software product.

Linking is when you link or join a FOSS component with your software product.

Modification is when you make changes to a FOSS component.

Translation is when you transform the code from one state to another.

When thinking about distribution of Open Source you should consider two things:
Who receives the software?

- Customer/Partner
- Community project

What is the format for delivery?

- Source code delivery

- Binary delivery
- Pre-loaded onto hardware

**Note was deleted.

CHAPTER 5

Running a FOSS Review

FOSS Review

- After Program and Product Management and Engineers have reviewed proposed FOSS components for usefulness and quality, a review of the rights and obligations associated with the use of the selected components should be initiated
- A key element to a FOSS Compliance Program is a *FOSS Review* process. This process is where a company can analyze the FOSS software it uses and understand its rights and obligations
- The FOSS Review process includes the following steps:
 - Gather relevant information
 - Analyze and **understand** license obligations
 - Provide guidance **compatible with** company policy and business objectives

The FOSS Review is a basic building block of a FOSS Compliance Program.

A FOSS Review can be the meeting point for engineering, business and legal teams, and can require planning and organization to successfully conduct on a large scale.

- Engineering or developer teams may participate in gathering relevant information
- Legal teams analyze and determine license obligations and provide guidance
- Business and engineering teams may receive and implement guidance

Initiating a FOSS Review



Anyone working with FOSS in the company should be able to initiate a FOSS Review, including Program or Product Managers, Engineers, and Legal.

Note: The process often starts when new FOSS-based software is selected by engineering or outside vendors.

The first step is to identify the proper parties to initiate a FOSS Review

Important questions to ask include:

- Who are the decision makers about FOSS usage (managers, architects, individual engineers, etc.)?
- How can they raise questions about FOSS usage?
- Is there a regular point in your development process where FOSS Reviews can begin?

What information do you need to gather?

When analyzing FOSS usage, collect information about the identity of the FOSS component, its origin, and how the FOSS component will be used. This may include:

- | | |
|---|---|
| <ul style="list-style-type: none"> • Package name • Status of the community around the package (activity, diverse membership, responsiveness) • Version • Download or source code URL • Copyright owner • License and License URL • Attribution and other notices and URLs • Description of modifications intended to be made | <ul style="list-style-type: none"> • List of dependencies • Intended use in your product • First product release that will include the package • Location where the source code will be maintained • Possible previous approvals in another context • If from an external vendor:  • Development team's point of contact • Copyright notices, attribution, source code for vendor modifications if needed to satisfy license obligations |
|---|---|

It should be noted that this list of information looks quite large. However, the amount of information required depends on the size of your company and what you intend to do with the FOSS code. Large entities tend to require more information than small entities.

There are a couple additional issues in the case of external vendors. First, you may need to follow up with the vendor if FOSS issues arise in the future, and having a reliable point of contact is important. You may also need to meet FOSS license obligations for FOSS delivered from the vendor. Ensure you have the notices and source code as needed to meet these obligations.

FOSS Review Team



A FOSS Review team includes the company representatives that support, guide, coordinate and review the use of FOSS. These representatives may include:

- Legal to identify and evaluate license obligations 
- Source code scanning and tooling support to help identify and track FOSS usage 
- Engineering Specialists working with business interests, commercial licensing, export compliance, etc., who may be impacted by FOSS usage

The FOSS Review team may consist of an interdisciplinary team

The legal team, which may include in-house or outside attorneys, reviews and evaluates the FOSS usage for license obligations

The legal team may be supported by others, including:

- Scanning and tooling teams that identify and track FOSS usage. These teams may provide support using code scanning or forensics tools to identify FOSS components in a codebase. The teams may also organize and track information gathered regarding FOSS usage to assist with later compliance processes.
- Other specialists or representatives that may be impacted by FOSS-related issues, such as commercial licensing, compliance or business planning teams.

Analyzing Proposed FOSS Usage



The FOSS Review team should assess the information it has gathered before providing guidance for issues. **This may include scanning the code to confirm the accuracy of the information.**

The FOSS Review team should consider:

- **Is the code and associated information complete, consistent and accurate?**
- Does the declared license match what is in the code files?
- Does the license permit use with other components of the software?



The FOSS Review team should have the expertise to properly assess the FOSS usage. This may require support from engineering teams to educate legal and business teams about the proposed FOSS usage. For example, code scanning may be used to locate undisclosed FOSS usage.

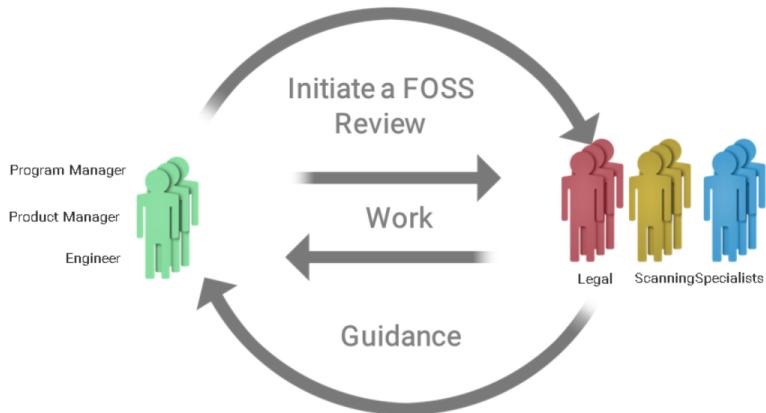
Once the proposed FOSS usage has been fully assessed, the legal team will then have the necessary information on which to make its judgments.

Source Code Scanning Tools

- There are many different automated source code scanning tools.
- All of the solutions address specific needs and - for that reason - none will solve all possible challenges
- Companies pick the solution most suited to their specific market area and product
- Many companies use both an automated tool and manual review
- A good example of freely available source code scanning tool is FOSSology, a project hosted by the Linux Foundation:
<https://www.fossology.org>

This slide explains the big picture of what Open Source code scanning tools are, how they work, and where a new user can start to gather knowledge about the subject.

Working through the FOSS Review



The FOSS Review process crosses disciplines, including engineering, business, and legal teams. It should be interactive to ensure all those groups correctly understand the issues and can create clear, shared guidance.

The FOSS Review process should be flexible enough to allow the interested parties to collaborate. Sometimes a FOSS usage scenario may not be clear to the FOSS review team. The engineering team will need the ability to provide further input. Likewise, the engineering team may need assistance in implementing guidance from the FOSS review team.

FOSS Review Oversight



The FOSS Review process should have **executive oversight to resolve disagreements and approve the most important decisions.**

The FOSS Review process should have oversight (for example, an Executive Review Committee in this diagram). The oversight committee may make important policy decisions or resolve disagreements between parties in the review process.

Check Your Understanding

- What is the purpose of a FOSS Review?
- What is the first action you should take if you want to use FOSS components?
- What should you do if you have a question about using FOSS?
- What kinds of information might you collect for a FOSS review?
- What information helps identify who is licensing the software?
- What additional information is important when reviewing a FOSS component from an outside vendor?
- What steps can be taken to assess the quality of information collected in a FOSS Review?

To gather and analyze information regarding FOSS usage and to produce appropriate guidance.

Initiate a FOSS review process. The method for initiating this process may vary by company, but should be open to those who are involved in using FOSS in development.

Initiate a FOSS review process or contact the FOSS review team. The process should be flexible enough so that FOSS users in your organization have access to guidance.

The package name, version, download URL, license, description and intended use in your product is a good starting point. The precisely level of detail you will need depends on your organization and intended use case.

The copyright notices, attribution and source code normally helps to identify who is licensing the FOSS software.

Development team's point of contact in case you need to follow up with future

FOSS issues. You may also want to obtain copyright and attribution notices, and source code for vendor modifications if these are needed to satisfy license obligations for FOSS licenses governing the third party software.

Check information for completeness, consistency and accuracy. This process may be assisted by support teams, including teams that run code scanning tools to scan for undisclosed FOSS usage.

****Note was deleted.**

CHAPTER 6

End to End Compliance Management (Example Processes)

Introduction

- Compliance management is a set of actions that manages OSS components used in products. Companies may have similar processes in place for proprietary components. FOSS components are called "Supplied Software" in the OpenChain specification.
- Such actions often include:
 - Identifying all the FOSS components used in Supplied Software
 - Identifying and tracking all obligations created by those components
 - Confirming that all obligations have been or will be met
- Small companies may use a simple checklist and enterprises a detailed process.



This slide describes the definition of compliance management and its end goals.

Note that this section provides a detailed example of what may take place in a large enterprise. Smaller companies may wish to approach the process in a more streamlined way.

**** This page was newly added.**

Example Small to Medium Company Checklist



Ongoing Compliance Tasks:

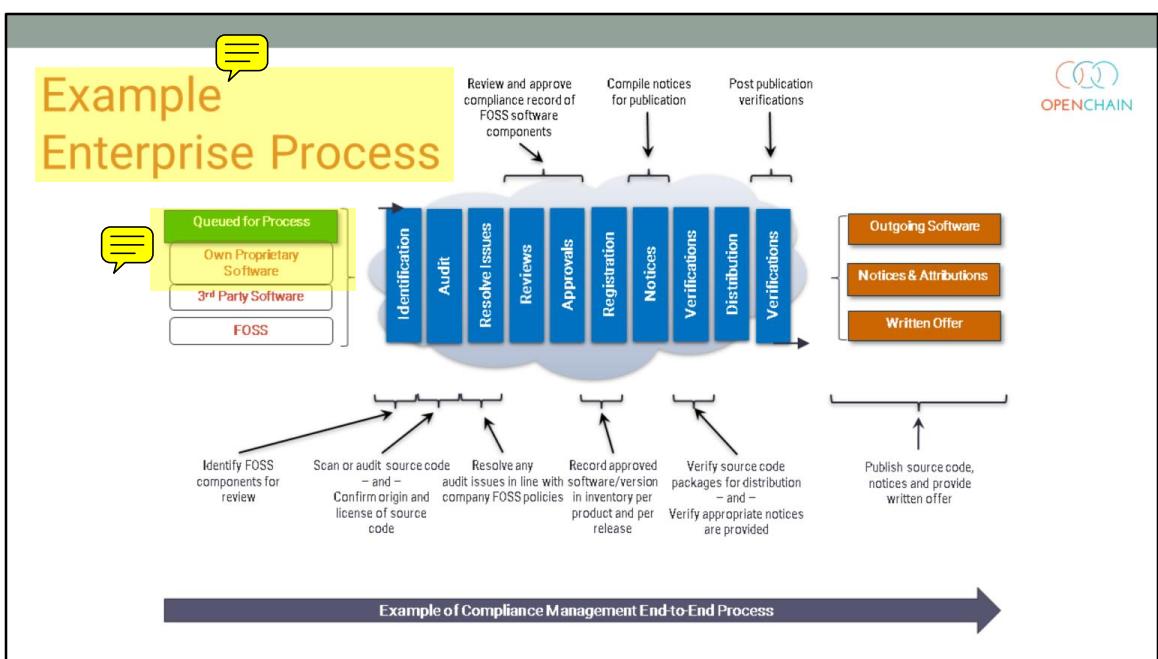
1. Discover all FOSS early in the procurement/development cycle
2. Review and Approve all FOSS components used
3. Verify the information necessary to satisfy FOSS obligations
4. Review and approve any outbound contributions to FOSS projects

Support Requirements:

1. Ensure adequate compliance staffing and designate clear lines of responsibility
2. Adapt existing Business Processes to support the FOSS compliance program
3. Have training on the organization's FOSS policy available to everyone
4. Track progress of all FOSS compliance activities

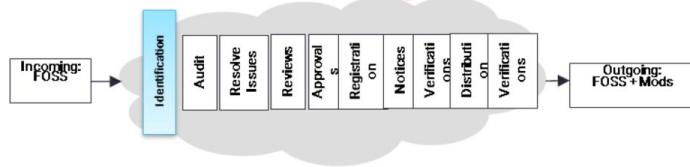
You can get detailed checklists for these items here: <https://www.linuxfoundation.org/projects/opencompliance/self-assessment-compliance-checklist>

This slide describes what a Small to Medium Enterprise (SME) might need to do to build and deploy an effective compliance program.



This slide is an overview of the steps that a larger enterprise might use for their process.

Identify and Track FOSS Usage



Identify FOSS components

- Steps:**

- Incoming requests from engineers
- Scans of the software
- Due diligence of 3rd-party software
- Manual recognition of new components added to the repository

- Outcome:**

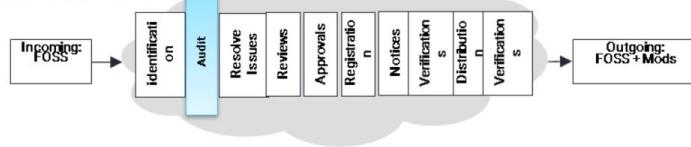
- A compliance record is created (or updated) for the FOSS
- An audit is requested to review the source code with a scope defined as exhaustive or limited according to FOSS policy requirements.

The first step in our example process is to identify FOSS usage.

This step may have been initiated by one of the events listed in “prerequisites.” For example, a development team may have initiated a request (or initiated a FOSS Review). The step may also begin if the review team discovers or is notified that FOSS is being used in a software release or in third party software used by the company, and that a proper review needs to take place.

In this example, the FOSS review team may identify FOSS usage through review requests from engineers, from performing scans of internally-developed and third-party software, or reviewing code checked into development branches. The review team will then create a record of the review, then move to the next step (“Audit”).

Auditing Source Code



Identify FOSS licenses

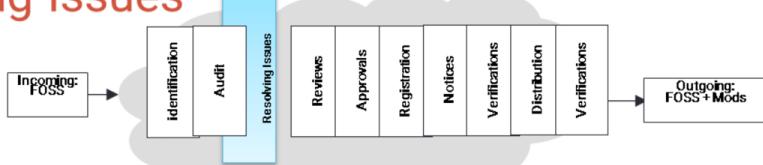
- **Steps:**
 - Source code for the audit is identified
 - Source may be scanned by a software tool
 - "Hits" from the audit or scan are reviewed and verified as to the proper origin of the code
 - Audits or scans are performed iteratively based on the software development and release lifecycles
- **Outcome:**
 - An audit report identifying:
 - 1. The origins and licenses of the source code
 - 2. Issues that need resolving

The next step is auditing source code identified in the previous step.

In our example, the company may conduct research into the identified FOSS component (e.g., review declared licenses, research origins of the FOSS component). The company may also scan the source code to verify the origin and composition of the code.

The review team may then produce an audit report with its conclusions regarding the origin and licensing of the source code.

Resolving Issues

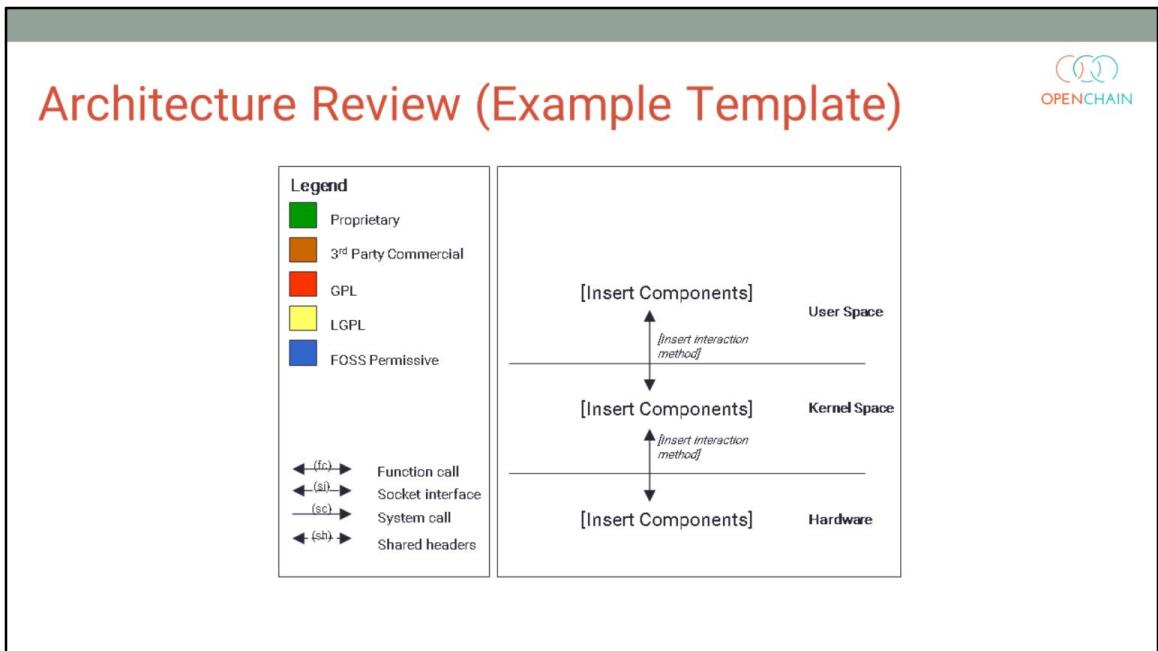


Resolve all issues identified in the audit

- **Steps:**
 - Provide feedback to the appropriate engineers to resolve issues in the audit report that conflict with your FOSS policy
 - The appropriate engineers then conduct FOSS Reviews on the relevant source code (see next slide for template)
- **Outcome:**
A resolution for each of the flagged files in the report and a resolution for any flagged license conflict

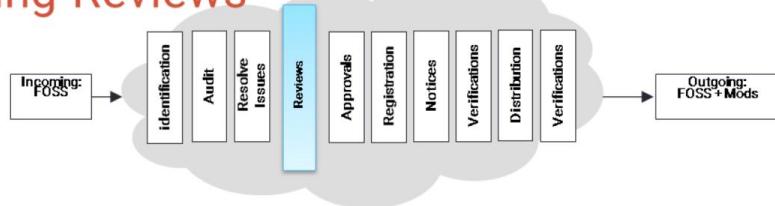
Once an audit report is produced that confirms the origin and licensing of source code, the review team should flag and review any issues under the company FOSS policy. For example, the earlier steps may have identified a FOSS component that contains other FOSS code under an incompatible license. The review team should provide appropriate feedback to the engineering team to resolve the issues.

** Sequential order of page changed.(contents is the same as before)



This slide contains a template that may be used to illustrate FOSS usage and its relationship with company software. For example, how are FOSS and company components linked together? Templates such as these may be created by engineering teams to help educate the FOSS review team about planned FOSS usage.

Performing Reviews



Review the resolved issues to confirm it matches your FOSS policy

Steps:

- Include appropriate authority levels in review staff
- Conduct review *with reference to your FOSS policy*

Outcome:

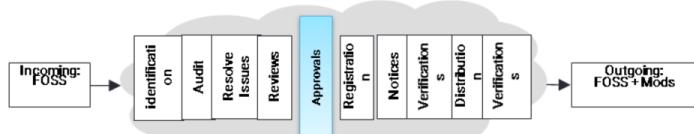
- Ensure the software in the audit report conforms with FOSS policies
- Preserve audit report findings and mark resolved issues as ready for the next step (i.e. Approval)

In this step, the FOSS review team reviews the facts collected in the previous steps and identifies the company's obligations under the FOSS licenses.

This step may be closely linked with the previous step (Resolving Audit Issues). In the previous step we removed FOSS usage that did not conform to company policy. In this step, we evaluate and identify the license obligations for FOSS usage that is retained.

Approvals

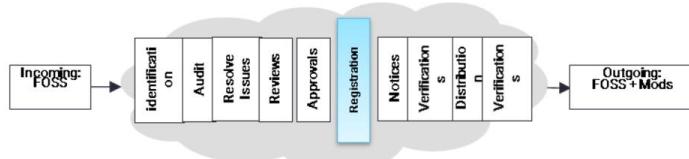
- Based on the results of the software audit and review in previous steps, software may or may not be approved for use
- The approval should specify versions of approved FOSS components, the approved usage model for the component, and any other applicable obligations under the FOSS license
- Approvals should be made at appropriate authority levels



In the approval step of our example process, the review team communicates whether it approves of the FOSS usage in question, along with any associated conditions or obligations. The approval should also include important details such as version numbers of FOSS components and the approved usage scenario.

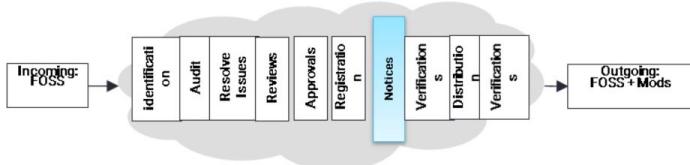
Registration / Approval Tracking

- Once a FOSS component has been approved for usage in a product, it should be added to the software inventory for that product
- The approval and its conditions should be registered in a tracking system
- The tracking system should make it clear that a new approval is needed for a new version of a FOSS component or if a new usage model is proposed



Approval information from the previous step should be tracked or registered so that anyone releasing the software can understand and comply with the relevant license obligations.

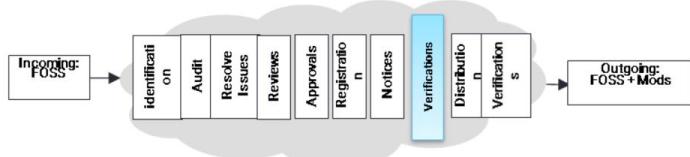
Notices



- Prepare appropriate notices for any FOSS used in a product release:
 - Acknowledge the use of FOSS by providing full copyright and attribution notices
 - Inform the end user of the product on how to obtain a copy of the FOSS source code (when applicable, for example in the case of GPL and LGPL)
 - Reproduce the entire text of the license agreements for the FOSS code included in the product as needed

If required by a FOSS license, appropriate notices should be prepared (often in a text file that accompanies the release). Notices may include attribution notices, modification notices, or offers for source code. For some licenses, you may also need to include a full copy of the license text.

Pre-Distribution Verifications



Verify that distributed software has been reviewed and approved

- **Steps:**
 - Verify FOSS packages destined for distribution have been identified and approved
 - Verify the reviewed source code matches the binary equivalents shipping in the product
 - Verify all appropriate notices have been included to inform end-users of their right to request source code for identified FOSS
 - Verify compliance with other identified obligations
- **Outcome:**
 - The distribution package contains only software that has been reviewed and approved
 - "Distributed Compliance Artifacts" (as defined in the OpenChain specification), including appropriate notice files are included in the distribution package or other delivery method



In this slide of our example process, the company verifies that it has met its FOSS license obligations before release. In cases where source code must be made available, the company verifies that the source code matches the binary files being distributed. The company also verifies that notices are properly produced and included in distribution packages as needed.

Accompanying Source Code Distribution

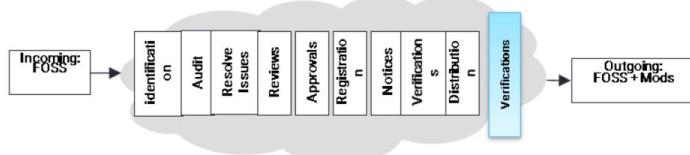


Provide accompanying source code as required

- **Steps:**
 - Provide accompanying source code along with any associated build tools and documentation (e.g., by uploading to a distribution website or including in the distribution package)
 - Accompanying source code is identified with labels as to which product and version to which it corresponds
- **Outcome:**
 - Obligations to provide accompanying source code are met

In cases where source code must be made available, the company provides the accompanying source code through the mechanisms permitted under the FOSS license. This may mean providing the source code along with the software distribution, making it available through a written offer, or posting a source code archive on a website.

Final Verifications



Validate compliance with license obligations

- **Steps:**
 - Verify accompanying source code (if any) has been uploaded or distributed correctly
 - Verify uploaded or distributed source code corresponds to the same version that was approved
 - Verify notices have been properly published and made available
 - Verify other identified obligations are met
- **Outcome:**
 - Verified Distributed Compliance Artifacts are appropriately provided

In this step, the company verifies that its distribution complies with its FOSS license obligations. This step could be a function of an entity providing oversight for the overall FOSS review process.

Check Your Understanding

- What is involved in compliance due diligence (for our example process, describe the steps at a high level)?
 - Identification
 - Audit source code
 - Resolving issues
 - Performing reviews
 - Approvals
 - Registration/approval tracking
 - Notices
 - Pre-distribution verifications
 - Accompanying source code distribution
 - Verification
- What does an architecture review look for?

For our example process, the steps include:

- Identification - Identify and track FOSS usage. This may take place through engineer requests, third party disclosures, or code scanning.
- Auditing source code - Review identified FOSS components for license and origin information.
- Resolving issues - Remove FOSS usage that is incompatible with FOSS policies.
- Performing reviews - Assess and determine obligations for FOSS usage.
- Approvals - Communicate approval conditions and license obligations.
- Registration/approval tracking – Track approval conditions and license obligations for later compliance steps.
- Notices - Prepare notices as required by FOSS licenses.
- Pre-distribution verifications – Review distributions for compliance before release.
- Accompanying Source Code Distribution – Make source code available as needed.
- Verification – Provide oversight for compliance process.

Architecture reviews examine the relationships between FOSS components and company software. For example, how are FOSS and company components linked together?

** Note was deleted.

CHAPTER 7

Avoiding Compliance Pitfalls

Compliance Pitfalls

This chapter will describe some potential pitfalls to avoid in the compliance process:

1. Intellectual Property (IP) pitfalls
2. License Compliance pitfalls
3. Compliance Process pitfalls

In this chapter, we will describe some common pitfalls to avoid in the FOSS compliance process.

Intellectual Property Pitfalls

Type & Description	Discovery	Avoidance
<p>Unplanned inclusion of copyleft FOSS into proprietary or 3rd party code:</p> <p>This type of failure occurs during the development process when engineers add FOSS code into source code that is intended to be proprietary in conflict with the FOSS policy.</p> 	<p>This type of failure can be discovered by scanning or auditing the source code for possible matches with:</p> <ul style="list-style-type: none"> • FOSS source code • Copyright notices <p>Automated source code scanning tools may be used for this purpose</p>	<p>This type of failure can be avoided by:</p> <ul style="list-style-type: none"> • Offering training to engineering staff about compliance issues, the different types of FOSS licenses and the implications of including FOSS in proprietary source code • Conducting regular source code scans or audits for all the source code in the build environment. 

The first pitfall described in this slide arises where copyleft-style licensed FOSS is inadvertently mixed with proprietary code.

This may be discovered through auditing source code for license notices or using code scanning tools.

Preventative measures include training of engineering staff, and building regular audits or scans into the development process.

Intellectual Property Pitfalls

Type & Description	Discovery	Avoidance
Unplanned linking of copyleft FOSS and proprietary source code:  This type of failure occurs as a result of linking software with conflicting or incompatible licenses. The legal effect of linking is subject to debate in the FOSS community.	This type of failure can be discovered using a dependency tracking tool that shows any linking between different software components.	This type of failure can be avoided by: 1. Offering training to engineering staff to avoid linking software components with licenses that conflict with your FOSS policies which will take a position on these legal risks 2. Continuously running the dependency tracking tool over your build environment
Inclusion of proprietary code into copyleft FOSS through source code modifications	This type of failure can be discovered using the audits or scans to identify and analyze the source code you introduced to the FOSS component.	This type of failures can be avoided by: 1. Offering training to engineering staff 2. Conducting regular code audits

The first pitfall in this slide arises where copyleft-style licensed FOSS is inadvertently linked to proprietary code.

This type of failure may be detected using dependency tracking tools or reviews of architecture.

Preventative measures include training of engineering staff, and building architectural reviews into the development process.

The second pitfall arises where proprietary code is included in copyleft-style licensed FOSS. For example, an engineering team making modifications to a FOSS component may include proprietary code in the modifications.

This type of failure may be discovered through auditing source code introduced into the FOSS component.

Preventative measures include training of engineering staff and building regular audits into the development process.

License Compliance Pitfalls

Type & Description	Avoidance
Failure to Provide Accompanying Source Code/appropriate license, attribution or notice information	This type of failure can be avoided by making source code capture and publishing a checklist item in the product release cycle before the product becomes available in the market place.
Providing the Incorrect Version of Accompanying Source Code	This type of failure can be avoided by adding a verification step into the compliance process to ensure that the accompanying source code for the binary version is being published.
Failure to Provide Accompanying Source Code for FOSS Component Modifications	This type of failure can be avoided by adding a verification step into the compliance process to ensure that source code for modifications are published, rather than only the original source code for the FOSS component

The first pitfall in this slide arises where a company has an obligation to provide accompanying source code, but fails to do so.

The second pitfall arises where a company provides accompanying source code, but fails to provide the correct version that matches the distributed binary version.

The third pitfall arises where a company modifies a FOSS component, but fails to publish the modified version of the source code. The company instead publishes the source code for the original version of the FOSS component.

In each case, the failures may be prevented by properly applying steps in the compliance process. For example, source code for released binaries should be captured and stored along with the binary version. Verifications prior to release should check to ensure the proper source code is provided with the binary release.

License Compliance Pitfalls

Type & Description	Avoidance
<p>Failure to mark FOSS Source Code Modifications:</p> <p>Failure to mark FOSS source code that has been changed as required by the FOSS license (or providing information about modifications which has an insufficient level of detail or clarity to satisfy the license)</p>	<p>This type of failure can be avoided by:</p> <ol style="list-style-type: none"> 1. Adding source code modification marking as a verification step before releasing the source code 2. Offering training to engineering staff to ensure they update copyright markings or license information of all FOSS or proprietary software that is going to be released to the public

The pitfall in this slide arises where a company modifies a FOSS component, then fails to mark its modifications when required by the FOSS license. This pitfall may be prevented through implementing processes for marking code or within verification steps.

Compliance Process Failures

Description	Avoidance	Prevention
Failure by developers to seek approval to use FOSS 	This type of failure can be avoided by offering training to Engineering staff on the company's FOSS policies and processes.	This type of failure can be prevented by: <ol style="list-style-type: none"> 1. Conducting periodic full scan for the software platform to detect any "undisclosed" FOSS usage 2. Offering training to engineering staff on the company's FOSS policies and processes 3. Including compliance in the employees performance review
Failure to take the FOSS training	This type of failure can be avoided by ensuring that the completion of the FOSS training is part of the employee's professional development plan and it is monitored for completion as part of the performance review	This type of failure can be prevented by mandating engineering staff to take the FOSS training by a specific date

The pitfalls in this slide arise from a failure to integrate the FOSS compliance process with the engineering team. In these cases, the engineering team does not raise FOSS usage to the review process, or does not receive the training on how to handle FOSS usage.

Preventative measures include monitoring of engineering training, and also making the compliance process easily accessible to the engineering team.

Compliance Process Failures

Description	Avoidance	Prevention
Failure to audit the source code	This type of failure can be avoided by: <ol style="list-style-type: none"> 1. Conducting periodic source code scans/audits 2. Ensuring that auditing is a milestone in the iterative development process 	This type of failure can be prevented by: <ol style="list-style-type: none"> 1. Providing proper staffing as to not fall behind in schedule 2. Enforcing periodic audits
Failure to resolve the audit findings (analyzing the "hits" reported by a scan tool or audit)	This type of failure can be avoided by not allowing a compliance ticket to be resolved (i.e. closed) if the audit report is not finalized.	This type of failure can be prevented by implementing blocks in approvals in the FOSS compliance process
Failure to seek review of FOSS in a timely manner	This type of failure can be avoided by initiating FOSS Review requests early even if engineering did not yet decide on the adoption of the FOSS source code	This type of failure can be prevented through education

This slide describes potential consequences of compliance process failures. In the first case, a code base may be used in development and releases without proper review. In the second case, FOSS usage may be known, but license obligations are not reviewed or determined. In the last case, the compliance process may face release deadline pressures and have limited time to perform its tasks.

Ensure Compliance Prior to Product Shipment

- Companies must make compliance a priority before any product (in whatever form) ships
- Prioritizing compliance promotes:
 - More effective use of FOSS within your organization
 - Better relations with the FOSS community and FOSS organizations

While avoiding the pitfalls described in this chapter may take resources and effort, prioritizing the FOSS compliance process is important. It can help you more effectively use FOSS in your development process, and also help maintain good working relationships within the FOSS community.

Establishing Community Relationships

As a company that uses FOSS in a commercial product, it is best to create and maintain a good relationship with the FOSS community - in particular, **with** the specific communities related to the FOSS projects you use and deploy in your commercial products.

In addition, good relationships with FOSS organizations can be very helpful in advising on best way to be compliant and also help out if you experience a compliance issue.

Good relationships with the software communities may also be helpful for two-way communication: upstreaming improvements and getting support from the software developers.

Your FOSS compliance process is a building block to establishing good working relationships within the FOSS community.

Check Your Understanding

- What types of pitfalls can occur in FOSS compliance?
- Give an example of an intellectual property failure.
- Give an example of a license compliance failure.
- Give an example of a compliance process failure.
- What are the benefits of prioritizing compliance?
- What are the benefits of maintaining a good community relationship?

Pitfalls can occur under the following categories: IP failure, license compliance failure, and compliance process failure.

An example of IP failure would be commingling of proprietary code and open source code, which may result in making proprietary software available to general public despite company's preference.

An example of license compliance failure would be a failure to mark an open source software after modification or to properly list the open source software components in the software or to make the complete and corresponding source code available.

An example of compliance process failure would be a failure in the process related to audit, review, or approving the open source software. Auditors "waived" all the red-flagged items in a report, or that the review and approval process takes too long.

The benefits of prioritizing compliance are that you become more efficient in your use of FOSS, and that you build a better relationship with the open source community.

The benefits of maintaining a good community relationship are that you can better

assess how you can comply with the FOSS license requirements, and you have a better two-way communication with regard to contribution and use of the FOSS.

** This Chapter was newly added (so from here, all slides were newly added.)

CHAPTER 8

Developer Guidelines

Developer Guidelines

- Select code from high quality, well supported FOSS communities
- Seek guidance
 - Request formal approval for each FOSS component you are using
 - Do not check un-reviewed code into any internal source tree
 - Request formal approval for outside contributions to FOSS projects
- Preserve existing licensing information
 - Do not remove or in any way disturb existing FOSS licensing copyrights or other licensing information from any FOSS components that you use. All copyright and licensing information is to remain intact in all FOSS components
 - Do not re-name FOSS components unless you are required to under the FOSS license (e.g., required renaming of modified versions)
- Gather and retain FOSS project information required for your FOSS review process

This slide outlines the key developer guidelines necessary for a high quality compliance approach.

Anticipate Compliance Process Requirements

- Include time required to follow established FOSS policy in work plans
 - Follow the developer guidelines for using FOSS software, particularly incorporating or linking FOSS code into proprietary or third party source code or vice versa
 - Review architecture plans and avoid mixing components governed by incompatible FOSS licenses
- Always update compliance verification - for every product
 - Verify compliance on a product-by-product basis: Just because a FOSS package is approved for use in one product does not necessarily mean it will be approved for use in a second product
- And for every upgrade to newer versions of FOSS
 - Ensure that each new version of the same FOSS component is reviewed and approved
 - When you upgrade the version of a FOSS package, make sure that the license of the new version is the same as the license of the older used version (license changes can occur between version upgrades)
 - If a FOSS project's license changes, ensure that compliance records are updated and that the new license does not create a conflict

This slide explains how to anticipate compliance process requirements.

Compliance Process Applies to all FOSS components



- In-bound software

- Take steps to understand what FOSS is included in software delivered by suppliers
- Evaluate your obligations for all of the software that will be included in your products
- Always audit source code you received from your software providers or alternatively make it a company policy that software providers must deliver you a source code audit report for any source code you receive

This slide emphasizes how a compliance process can and should apply to all FOSS components entering your company.

Check Your Understanding

- Name some general guidelines developers can practice when working with FOSS.
- Should you remove or alter FOSS license header information?
- Name some important steps in a compliance process.
- How can a new version of a previously-reviewed FOSS component create new compliance issues?
- What risks should you address with in-bound software?

Learn more through the free Compliance Basics for Developers hosted by the Linux Foundation at:

<https://training.linuxfoundation.org/linux-courses/open-source-compliance-courses/compliance-basics-for-developers>

General guidelines developers can practices when working with FOSS:

- Select code from high quality FOSS communities
- Seek guidance
- Preserve existing licensing information
- Gather and retain FOSS project information for your review process

Should you remove or alter FOSS license header information? No – existing

license information should be preserved, additional header information can be added for modifications or additions to source code (note, some licenses require documenting changes) .

Important steps in a compliance process:

- Follow developer guidelines, especially for any FOSS code included in or linked to proprietary code
- Review and approve all FOSS early in the cycle
- Review architecture and avoid mixing components governed by incompatible licenses
- Verify OSS compliance for every product and every version prior to release
- Review OSS compliance for new versions of OSS

A new version of a previously reviewed FOSS component can create new compliance issues by:

- A change in the FOSS license for the new version of the FOSS component(e.g. ghostscript <https://en.wikipedia.org/wiki/Ghostscript>)
- New dependencies introduced with new versions which create additional FOSS obligations. These dependencies may be embedded in the FOSS distribution or they may be dependencies resolved at build time.

What risks should you address with in-bound software?

- License compliance for any disclosed FOSS embedded in the in-bound software
- The potential for creating license conflicts by integrating inbound software with other FOSS or proprietary software
- Undisclosed or unknown FOSS included in the in-bound software