Open Source Software Paradigms

Lecture - 10

Legal Aspects of Open Source (Licenses & Copyrights)

Lecturer: Biniam Behailu
Addis Ababa Science and Technology University

Learning Objectives

By the end of this lecture, you will be able to:

- Define the core intellectual property rights—copyright, patent, and trademark—and explain their role in open source software.
- Explain how copyright forms the legal backbone of open source licensing.
- Differentiate between Contributor License Agreements (CLAs) and the Developer Certificate of Origin (DCO), and describe their purposes.
- Identify common open source compliance obligations and the consequences of non-compliance.
- Analyze the business and legal rationale behind proprietary relicensing and dual-licensing models.

Contents

- ☞ Intellectual Property Foundations
- ☞ The Collaboration Framework
- ☞ The Rule of Law: Compliance
- ☞ Business & Licensing Strategies
- ☞ Emerging Frontiers & Enforcement

Intellectual Property

What is Intellectual Property ?

- Intellectual property is a legal right that grants the creator of original work exclusive rights to its use and distribution.
- It is the umbrella term for creations of the mind that the law protects.
- **It includes:**

☞ Copyrights, Patents, Trademarks, Trade secrets, Industrial designs, etc.

***Image source**: "Intellectual Property," sheikhco.ir, [Online]. Available: https://sheikhco.ir/wp-content/uploads/2025/04/intellectualproperty_vyapaarJagat.webp. [Accessed: 19-Oct-2025].*

What is Intellectual Property ?

- Open source projects are built on IP law, not outside it.
- The **creators** own the **copyright**, **patents**, and **trademarks**, they just choose to license those rights under terms that permit others to use, modify, and share the work.

> Open source = *"We own it, but we give you permission to use it under certain conditions."*

Copyright

- Copyright is a legal right that grants the creator of original work exclusive rights to its use and distribution.
- Copyright automatically applies to **original** works of authorship, including software code, the moment they are "fixed in a tangible medium."
- It grants the author exclusive rights: to **reproduce**, **distribute**, **create derivative works**, and **publicly** perform/display.
- Without a license, no one else has these rights.
- Using software without a license is **copyright infringement**.

Copyright

Copyright is the backbone of open source licensing.

- Every contributor automatically owns copyright over what they create.
- The open-source license (like MIT, GPL, Apache, BSD) is the legal mechanism that grants permissions for use, modification, and redistribution.
- Without copyright, there's no legal basis for licensing at all.

> **Example**
>
> If you publish code under the MIT License, you're saying:
>
> *" I, the copyright holder, give anyone permission to use, modify, and distribute this code but they must include my copyright notice. "*

Copyright

- The copyright holder has the exclusive right to do, or to authorize others to do, the following:
- **Reproduction**: Making copies of the code.
- **Derivative Works**: Creating new works based on the original (modifying or adapting the code).
- **Distribution**: Selling or otherwise transferring copies to the public.

- **Note**: An Open Source License is simply the copyright holder giving users permission to exercise some or all of these rights under specified conditions.

Patent

- Patent is a form of intellectual property that gives the patent holder exclusive rights to an invention for a certain period.
- It protects new, useful, and non-obvious inventions or processes.
- Even if source code is open, using it could infringe a patent if not explicitly licensed.

Example

- Apache License 2.0 → includes an explicit patent license.
- GPLv3 → includes a strong patent retaliation clause (if you sue for patent infringement, you lose your license).

Trademark

- Trademark is a symbol, word, or phrase legally registered to represent a company or product.
- It protect names, logos, and brands, not code.

Example

- You can use Firefox’s open-source code, but you can’t call your fork “Firefox” without permission.
- Red Hat allows access to open-source software but restricts use of the Red Hat name and logos.

Trademark

- Trademark control allows companies to:
  - ☞ Maintain quality assurance in open ecosystems.
  - ☞ Prevent confusion between official and unofficial versions.

Contributor Agreements

The Problem: Intellectual Property Chain of Title

- A project is a combination of code from many contributors.
- To distribute the project under a single license, the project must have the legal right to use everyone's contributions.

> **The Risk**
>
> A contributor could later claim,
>
> *“I never gave you a license to my code," invalidating the project's license and creating massive liability. ”*

Contributor agreements

- Contributor agreements are crucial legal documents in (OSS) ecosystem.
- Essential for managing contributions to a project.
- To ensure **clarity** and **legal** protection for both the **contributors** and the **maintainers** of the project.
- For intellectual property management purpose and to encourage contributions.

Contributor agreements

- There are three ways to handle copyright ownership for free code and documentation contributed by multiple individuals

  1. Ignore the Issue of Copyright Entirely
  2. Collect a Contributor License Agreement (CLA)
  3. Obtain Actual Copyright Assignments (CA)

What is a Contributor License Agreement (CLA)?

- A legal agreement between a contributor and a project/project owner.
- It formally outlines the terms under which the contribution is made.
- It is separate from the project's Open Source license.
- It is about the inbound code, not the outbound distribution.

Why are CLAs Critical?

- **Patent Protection -** Ensures the contributor grants an explicit patent license for their contribution.
- **Relicensing Flexibility -** Gives the project entity the right to relicense the code (e.g., for a dual-license model).
- **Legal Defense -** Allows the project entity to legally represent the entire codebase in enforcement or defense actions.
- **Clarity** & **Certainty -** Provides a clear chain of authority for all code in the project.

Anatomy of a CLA

A typical CLA will specify:

- **Grant of Copyright License -** Contributor grants a perpetual, worldwide, non-exclusive, no-charge, royalty-free copyright license.
- **Grant of Patent License -** Contributor grants a patent license for any patents their contribution infringes.
- **Representations** - Contributor confirms they have the rights to contribute.
- **No Obligation -** The project is not obligated to use the contribution.

Why Do Projects Use CLAs?

1. IP Integrity & Defense:

- Ensures the project has a clear, defensible right to all its code.
- Allows the project to confidently enforce its license.
- Enables the project to relicense the code in the future if needed.

2. Patent Protection:

- Explicit patent grants protect all downstream users from patent claims by contributors.

CLA vs. Copyright Assignment

Copyright Assignment

- Contributor transfers ownership of their copyright to a single entity (e.g., FSF, Project Foundation).
- The foundation becomes the sole copyright holder.
- More control, but a bigger "ask" from contributors.

Contributor License Agreement

- Contributor keeps ownership but grants a broad license.
- Easier for contributors to sign, still provides most key benefits.

The Developer Certificate of Origin (DCO)

- A lightweight alternative to a CLA.
- Used by the Linux kernel, Node.js, Eclipse.
- It is a assertion made via a Signed-off-by line in a git commit.
- The developer certifies they have the right to submit the code under the project's license.
- Does NOT grant relicensing rights. Simpler, but offers less flexibility.

CLA vs. DCO (Developer Certificate of Origin)

- **CLA**: A formal, often lengthy, legal agreement. Common in corporate projects (Google, Apache Foundation).
- **DCO**: A lightweight alternative. A signed commit message where the developer certifies they have the right to submit the code under the project's license.
- **Philosophy**: DCO is simpler and places less burden on developers; CLA provides stronger legal certainty for the project owner.

The Rule of Law (Compliance)

What is Open Source Compliance?

- The process of meeting all the obligations defined in the licenses of the Open Source software you use.
- It's not about "if" you comply, but "how" you prove it.
- **Goal**: Use the power of OSS without exposing your organization to legal risk.

The Top 5 Compliance Failures

1. Failure to Provide Source Code (GPL violation).
2. Failure to Include License Notices & Attribution (all licenses).
3. Misunderstanding "Distribution" (Internal vs. External, SaaS vs. on-prem).
4. Ignoring Copyleft Propagation when linking libraries.
5. License Incompatibility when combining OSS packages.

Why Compliance Matters: The Consequences of Failure

- **Copyright Infringement Lawsuits**: (e.g., Artifex v. Hancom)
- **Injunctions**: Halting distribution of your product.
- **Financial Damages**: Legal fees, fines, and lost revenue.
- **Reputational Harm**: Loss of trust in the developer community.
- **Forced Source Code Release**: A court may order you to comply with the license terms, making your proprietary code public.

Common Compliance Obligations

1. **Attribution**: Including copyright notices in documentation and "About" boxes.
2. **License Text**: Providing a copy of the license itself.
3. **Source Code Offer**: For GPL, providing the complete corresponding source code for any distributed binaries.
4. **Modification Notice**: Documenting changes made to the OSS code.
5. **Patent Notice**: Preserving Apache 2.0 patent notices.

The Compliance Process: A 5-Step Framework

1. **Discover**: Identify all OSS in your codebase. (SBOM)
2. **Analyze**: Determine the licenses and their obligations.
3. **Plan**: Decide how to fulfill each obligation (notices, source delivery).
4. **Execute**: Implement the plan in your product distribution.
5. **Audit/Monitor**: Continuously scan and review, especially for new dependencies.

Tools for Compliance

- **Software Bill of Materials (SBOM):** A nested inventory of all software components.
- **Scanning Tools**: FOSSology, Scancode-Toolkit, Black Duck, Snyk.
- **Automation**: Integrate scanning into your CI/CD pipeline to catch issues early.

Compliance Pitfall: "Distribution" is the Trigger

- Copyleft obligations are triggered when you distribute software externally.
- **Internal Use?** Generally, no distribution, so no Copyleft source disclosure required.
- **SaaS / Network Service?** Generally not a "distribution" under GPL, but it is under AGPL.

Compliance Pitfall: License Incompatibility

- You cannot combine code from two licenses if their terms conflict.
- **Example**: GPLv2 is incompatible with Apache 2.0.
  - ☞ GPLv2 says: "All code must be under GPLv2."
  - ☞ Apache 2.0 has additional patent terms that are "further restrictions" under GPLv2.
- **Solution**: Avoid mixing them, or use a GPLv2-compatible license like GPLv3 or MIT.

Proprietary Relicensing

What is Proprietary Relicensing?

- Proprietary relicensing is when the owner of intellectual property (like software, data, or creative work) changes its licensing terms to a proprietary model, meaning it’s no longer freely available under open or permissive licenses.

- This transition can be seen as a betrayal of the open-source community and can have significant implications for users and developers.

- Users who initially received the software under an open-source license may lose their rights to use it if the software is re-licensed.

Why Do Companies Relicense?

- **Commercialization**: The company may want to monetize the software by selling it or offering proprietary features and support.
- **Strategic Shift**: The company may decide to focus on a different business model or product strategy.
- **Control**: Relicensing allows the company to have more control over the software's development and distribution.
- **Intellectual Property Protection**: The company may want to protect its intellectual property and prevent others from using it without permission.

Common Scenarios

1. Open Source → Proprietary

- ☞ A company decides to stop releasing updates publicly.
- ☞ **Example**: A firm takes its open-source software and starts selling it as a commercial product under a closed license.

2. Dual Licensing

- ☞ The product is released under both open-source and proprietary licenses.
- ☞ **Example**: MySQL and Qt have done this — open source for community users, proprietary for commercial users.

3. Proprietary → Proprietary (Relicensing with New Terms)

- ☞ Changing the licensing model (e.g., from perpetual to subscription).
- ☞ Often happens when companies shift business models, merge, or get acquired.

💡 Example

- A startup originally releases its software under the MIT License. Later, after gaining traction, it relicenses it under a proprietary EULA to monetize it commercially. The old MIT-licensed versions remain open, but new ones are closed.

> **Relicensing note**
>
> When a project wants to relicense, all copyright holders (contributors) must agree, unless contributor agreements give that power to a maintainer or company.

How is Proprietary Relicensing Possible?

- **It Requires 100% IP Control:** The copyright holder(s) must own or have rights to all the code/content before relicensing.
- The licensor must own the copyright to every single line of code in the project.
- If there are multiple contributors, you need their explicit consent unless a Contributor License Agreement (CLA) already grants relicensing rights.
- The CLA gives the project owner the legal right to offer the software under a different license (the proprietary one) without needing permission from every contributor.

Case Study: MySQL

- **OSS License**: GPLv2.
- **Commercial License**: Proprietary, for a fee.
- **Success Story**: A company built a proprietary product embedding MySQL? They had to either open-source their product or buy a commercial license. This created a massive revenue stream, leading to a $1 billion acquisition by Sun Microsystems.

***Image source**: [1] "MySQL Logo," 1000logos.net, [Online]. Available: https://1000logos.net/wp-content/uploads/2020/08/MySQL-Logo-640x400.png. [Accessed: 07-Oct-2025].*

The Modern Evolution: Source-Available Licenses

- **Examples**: Server Side Public License (SSPL) (created by MongoDB), Elastic License, BSL (Business Source License).
- These are not Open Source (OSI-approved).
- **Model**: The source code is visible, but the license explicitly prohibits commercial SaaS offerings or use by large competitors.
- **Goal**: Achieves the same outcome as dual licensing without maintaining two separate licenses.

The AI Frontier: Training & Output

- **Training on OSS**: Is using GPL code to train an AI "fair use" or creation of a derivative work? (Unsettled Law).
- **Output Compliance**: If an AI generates code snippet identical to its GPL-trained data, is the user liable?
- **Best Practice**: Treat AI-generated code as high-risk third-party code and scan it rigorously.

The Enforcement Landscape

- **Who Enforces?** Copyright holders (individuals, FSF, Software Freedom Conservancy).
- **How?** Often start with a private violation notice, escalating to litigation.
- **Trend**: Increased enforcement by commercial actors protecting their dual-licensed or source-available products.

Summary

- IP is the Foundation
- Licenses are Permissions
- CLAs Enable Scale
- Compliance is Mandatory
- Licensing is a Business Strategy
- The Landscape is Evolving

Brain Teaser

1. What is the fundamental legal mechanism that makes open source licensing possible?

A. Trademark Law

B. Patent Law

C. Copyright Law

D. Trade Secret Law

Brain Teaser

1. What is the fundamental legal mechanism that makes open source licensing possible?

A. Trademark Law

B. Patent Law

C. Copyright Law

D. Trade Secret Law

Brain Teaser

2. What is the primary purpose of a Contributor License Agreement (CLA)?

A. To set the price for using the software

B. To manage the inbound rights to contributions and protect the project

C. To manage the inbound rights to contributions and protect the project

D. To assign all copyright from the contributor to the project

Brain Teaser

2. What is the primary purpose of a Contributor License Agreement (CLA)?

A. To set the price for using the software

B. To manage the inbound rights to contributions and protect the project

C. To manage the inbound rights to contributions and protect the project

D. To assign all copyright from the contributor to the project

Thank you!

"Open source software is a testament to the power of collaboration; it transforms ideas into innovations, empowering individuals and communities to build a better future together."

Lecturer: Biniam Behailu
Addis Ababa Science and Technology University