9.Licences: proprietary, free and open-source

Open-source licenses

While many different open-source licenses exist, their requirements fall under two broad categories: “permissive” and “copyleft.”

Permissive

A permissive license allows developers and organizations to use, modify, and distribute software with few restrictions. These licenses are a popular choice among organizations looking to sell software. They enable developers to create and innovate while allowing organizations to keep their software proprietary and monetize it.

However, it's important to note that although the term 'permissive' suggests leniency, failure to comply with the legal obligations of these licenses can still result in serious consequences. These include expensive lawsuits, publication of proprietary software, and loss of business.

Examples of permissive licenses include the Berkeley Software Distribution (BSD) and the Massachusetts Institute of Technology (MIT) license. BSD licenses were originally 4-clause licenses; however, BSD licenses can now have 0–4 clauses. The MIT License was originally developed for X Window System in the 1980s, and is now one of the most popular open source software (OSS) licenses available.

While these licenses more generally address “attribution” or giving the author credit for their work, some have no legal obligations. These are the “zero clause” licenses, including the BSD 0-clause and MIT “No Attribution,” which behave the same as those in the public domain.

Public domain

Public domain OSS is free to use and is not owned by any organization. While anyone can use the software in the public domain, users assume all risks of the software. So it's important to continue other security measures.

Examples of public domain software licenses include CC0 (Creative Commons Zero).

Copyleft and weak copyleft

A protective license with clear restrictions, copyleft requires anyone distributing the software to make the source code available. A reciprocal license is another term that describes copyleft licenses like the General Public License (GPL) or Affero General Public License (AGPL). This is because Copyleft licenses like GPL require any derivative works or modifications to be released under the same license as the original software.

Anyone who uses or modifies software with a GPL license must also share their changes and improvements with the wider community. Put simply, a GPL license can spawn more GPL licenses. It is important to note that the reciprocal nature of these licenses is intentional and aims to promote the sharing and free distribution of software.

The GPL's concept of sharing software code changes has been suggested as a driving force behind the success of Linux. However, because numerous organizations have included GPL-licensed software commercial hardware and software, they have been required to share intellectual property, including source code. As a result, Sonatype's license policy standards has pre-set the strictest of these licenses as "Banned" so components containing strong copyleft licenses will not make it into production.

Within the copyleft license category, there is a so-called weak copyleft license. These licenses have fewer restrictions, allowing the linking of original software without requiring your software to be released under the same license. While the changes to the original software must still use the same license, the rest of the software can be released under a different license. This makes using the original software in proprietary projects easier than standard copyleft.

Weak copyleft licenses include the Mozilla Public License used by Mozilla and LibreOffice and the LGPL used by projects like FFMPEG.

Non-standard

These licenses may fall under permissive or copyleft, depending on their restrictions. Non-standard licenses are distinct from other open source licenses due to unconventional and often downright funky terms of use. The strange terms of these licenses can put some of them into a legal gray zone and are often impractical.

For example, Beerware is a term used to describe a software license based on the "beer-buying" tradition in some cultures. This means if you ever meet in person, you need to buy the owner of the component a beer. Oftentimes these licenses, like the WTFPL, aim to poke fun at other copyleft or permissive licenses.

Top 5 examples of open source software

There are several open source software solutions, each serving a purpose. Listed below are some examples of open source software

VLC media player: VLC media player is one of the most popular media players we have today. It supports most video and audio files, including MOV and MP3. It also supports streaming.

Mozilla Firefox: Firefox is one example of an internet browser that offers a free and safe browsing experience. Firefox has similar features to Opera Mini and Chrome browser and has features that protect you while browsing. It's customizable and supports browser extensions.

WordPress: WordPress is a content management system (CMS) and website builder that allows you to create and store website content. It has very basic features, which you can extend by installing plugins and themes. You can download and install a standalone version on your server, or you can buy a managed hosting with WordPress already installed from WordPress.

React: Reactjs, a front-end JavaScript library allows you to build single-page applications (SPA). React allows you to break your application's user interface (UI) into reusable server or client components.

TestDisk: TestDisk is a file recovery software that can be used to recover lost partitions on your PC. TestDisk can recover files from different file systems including NTFS, exFAT, and FATX. TestDisks also works on other storage devices including USB drives and memory chips.

Types of open source software

Depending on the restrictions and rules for collaboration on software, an open source license can be considered as either permissive or copyleft.

Permissive license: Permissive licenses are the less restrictive type of open source software license. Permissive licenses allow anyone to freely modify and share your software, use your source codes as part of their software, and distribute it in proprietary works. Often, they only require you to provide attribution to the original developers when distributing the software.

Examples of permissive licenses include the Massachusetts Institute of Technology (MIT) license, the Berkeley Source Distribution (BSD) license, and the Apache License.

Copyleft license: Unlike permissive licenses, copyleft licenses are very restrictive. Copyleft licenses require anyone distributing software that contains source codes protected under a copyleft license to do so under the copyleft terms.

Copyleft licenses intend to make the source codes of modified versions of software available to the public to prevent being used in proprietary works without proper attribution.

Examples of copyleft licenses include General Public License(GPL), Affero General Public License(AGPL), and Mozilla Public License(MPL).

Does open source mean free of cost?

Open source software is usually distributed for free. However, additional features and services may come at a cost. Commercial Open Source Software companies have developed business models that help them commercialize free software. These models often revolve around selling support or hosting or selling add-on features to complement the free software. Software run under

any one of these business models is referred to as Commercial Open Source Software (COSS).

Commercial Open Source Software examples include WordPress, Unreal Engine, and MongoDB.

Advantages of choosing an open source license

Choosing an open source license has several advantages. Some of these are:

Community collaboration: Choosing an open source license invites a global community of developers, designers, and users to collaborate on your project. You get to improve your software and fix bugs for free.

Rapid iteration: With a larger pool of contributors, development cycles can become faster. Bugs are identified and fixed quickly, new features are proposed and implemented, and your project can evolve more rapidly than you could have imagined.

Quality improvement: Changes to open source software are often peer-reviewed. The scrutiny of the open source community can lead to higher code quality. Contributors review the code base thoroughly to ensure best practices, identify vulnerabilities, and enhance overall reliability.

Innovation: Open source fuels many modern-day inventions. Many technologies we rely on today are open source. Such an example is the internet. Choosing an open source license allows anyone to take your original idea and make something new from it.

Mass adoption: According to GitHub's octoverse 2022 report, 90% of companies rely on open source software. Making your software open source means you are tapping into the population of businesses already using open source software.

Disadvantages of choosing an open source license.

Let's look at some disadvantages of choosing an open source license for your project.

Limited support for users: Open source software often lacks dedicated support teams to help users resolve issues with the software. Contributors are usually more interested in building and shipping new features to users than in

supporting users to resolve issues they encounter while using the software. Often open source software users would need to rely on discussion forums like stackoverflow to resolve an issue.

Bad documentation: Open source software documentation often receives less attention. The software documentation is usually written by the community of developers working on the projects. Sometimes, the software documentation is adapted for users with technical knowledge and may be difficult to understand as a normal user with less technical knowledge.

Security issues: Attackers can learn and find vulnerabilities in open software much easier compared to closed source software. Sometimes, the vulnerability might come from your software dependencies, which are exposed to attackers. In other cases, some developers might contribute bugs to your software to make it vulnerable and easy to exploit.

Limited funds: Oftentimes, free open source projects not backed by big companies rely on crowdfunding or donations. With limited funds, it can be hard to invest in further development of your software.

Project abandonment: Open source software contributors are more likely to abandon your software for other open source software, and it can become challenging to find new contributors for software whose core developers have stopped working on it.

Commercial software examples include:

Adobe Photoshop: Photoshop is a photo-editing software that allows you to edit and save your photos and graphics. It offers similar features to other photo-editing software like Figma and Gimp and supports feature extensions through plug-ins.

DigitalOcean: Digital Ocean is an example of a commercial open source cloud service provider. Digital Ocean offers cloud computing and lets you host your website and applications back-end on a cloud infrastructure.

Wondershare Filmora: Filmora is a video editing software that allows you to edit videos and audio. Filmora offers similar tools to other video editing software like Adobe After Effects. Filmora also supports video export in different file formats including MOV, 3GP, and MP4.

Bigcommerce: Bigcommerce is an example of an open source Software as a Service(SaaS) ecommerce provider. Bigcommerce provides retailers tools to set up an online store without much hassle.

Zoom: Zoom is a virtual meeting software that offers video, audio, and messaging tools to communicate effectively with others over the internet. Zoom also offers other features like meeting transcription and virtual whiteboard as part of their software.

Types of commercial software licenses

Commercial (or proprietary) software licenses come in various types, each with its terms and conditions set by the software vendor or developer. Here are some common types of proprietary software licenses:

Single-user license: As the name implies, single-user licenses allow a single person to use one installed copy. This means other users of the software need separate copies of the software license for themselves.

Volume license: Volume licensing is suitable for organizations that need multiple copies of the software. These licenses allow you to share copies of software in the organization using only one license.

Perpetual license: A perpetual license grants the right to use the software indefinitely, usually with the option to purchase maintenance and support separately.

Subscription or annual license: With these types of software, you purchase a license that grants you access to a copy of the software for a particular period (often a year), after which you need to renew your software license if you wish to continue using the software.

Floating or concurrent license: Floating licenses allow a specified number of users to access the software on a network. These types of licenses are managed by a license server. The license server tracks and maintains the specified number of users using the software simultaneously.

Advantages of choosing a commercial licensing model

Several benefits come with commercial license models. Commercial licenses give you flexibility and control over your software. Let’s discuss some reasons why you should consider commercial license models

Protects interest: Choosing a commercial license for your proprietary works protects your interest in the software. As mentioned earlier, commercial software licenses protect your business and preserve your rights. These licenses may include clauses to restrict certain activities, such as reverse engineering your software and redistributing copies of your proprietary works.

Maintains ownership: Commercial license models often do not license ownership or the rights to modify and distribute a software copy to the end user. Commercial software licenses usually restrict others from using your source code.

Maintains competitiveness: Distributing your software under commercial licenses gives you a competitive advantage over open source software. Some users prefer licensed software over open source software for several reasons, including security and support.

Maintains control: Licensing your software to users gives you control over your software. Simply put, you control who gets access to your source code and who can work on your software. To an extent, you can also control how the end user uses your software.

Develops funds: Licensing your software to end users generates revenue used to fund further research and developments. Funding gives you the advantage of employing people to help develop features and improve the software.

Disadvantages of choosing a commercial licensing model.

While licensing your software to end-users might be lucrative, it poses certain disadvantages. Let’s discuss some disadvantages of choosing commercial licensing models

Impending liability: Commercial software owners or companies are often liable for any damage caused by defects in their software. They’re responsible for the data protection and privacy of their users and may be subjected to litigation if any issues arise.

Software piracy: Commercial software is often pirated by users who do not want to pay for a license. Such activities affect your ability to raise funds from your software. Piracy could also harm your brand’s identity in the long run.

Manufacturer dependence: Commercial software usually offers little customization options for users. As a result, users tend to depend on the

software manufacturer to fix bugs in the released software and add features they need in further updates.

High costs: Commercial software can be expensive to build, maintain, and scale. Commercial software owners would usually need some funds upfront to build fully functional software for their target users.

Slower development cycle: Commercial software projects are often developed by a small number of developers. With fewer people working on the software your development cycle would be much slower compared to a similar but open source software.

Open source and proprietary software similarities

Similarities between open source and proprietary software include:

Product documentation: Both open source and commercial software are distributed with documentation to help end users complete tasks using the software.

Skilled developers: Open source and commercial software are developed and maintained by a community of skilled developers.

Customer support: Both open source and commercial software may have technical support teams to help users troubleshoot and resolve issues relating to the software.

Security concerns: Open source and proprietary software are both vulnerable to hacking. As a result, individual developers or companies invest time and effort to fix vulnerabilities in their software.

Compiled versions: Open source and proprietary software are typically distributed in compiled form. However, contrary to proprietary software, open source software source codes are made publicly available.

Copyrighted: Both open source and proprietary licenses are subject to copyright laws.

1. Open-source software vs Free software vs Freeware

The ground notion is that “open-source software”, “free software”, “freeware” and “public domain” are different concepts. These definitions do NOT refer to the types of the software, but to the values lying underneath. They might intersect in various manners and refer to the same set of licenses. For example, the majority of free software licenses are also considered open-source software licenses, like well-known Linux, Ubuntu, MySQL, to name a few.

Open-source license criteria focus on the availability of the source code and the ability to modify and share it, while free software and public domain focus on the user’s freedom to use the program, to modify it, and to share it. Freeware (examples are Skype and Adobe Acrobat), in its term, is mostly aiming commercial goals and potential monetization often used as a “freemium” product.

2. Copyright vs Copyleft vs Permissive vs Creative Common

These terms are used to compare legal attributes of open-source and free software and other content publicly available licensing to proprietary licenses.

Permissive licenses place minimal restrictions on software users. Often they only require that the original creators are attributed in any distribution or derivative of the software or source code.

For example, the software under permissive license may be incorporated into the other proprietary software without disclosing the source code, and this newly created software may be distributed. Because of their origins, permissive open-source licenses are sometimes called “academic” licenses and frequently used by academic institutions.

For software creators who want to guarantee unlimited open-source access to their work employ the concept of copyleft. Copyleft uses copyright's legal framework to ensure continued open access to software and its source code. The core requirement that any derivative work must be distributed under the same licensing terms as the original. Because of this requirement, copyleft licenses are considered "restrictive" licenses, though these restrictions guarantee open access.

Creative common is most commonly applied to various content that used along with software i.e. images, videos, texts, etc. There are 6 Creative common licenses, which we will not cover in this article.

3. MIT vs BSD vs Apache vs GPL vs LGPL vs AGPL

Although there is an enormous amount of various licenses on the market, we will cover only a few most popular ones, which we at Moqod frequently use for developing projects for our clients.

Here are the most popular open-source licenses in a simplified comparative table and a brief outline one by one.

Apache 2.0 is the most modern and balanced among permissive licenses. It is written clearly regarding the modern copyright usage rules, in particular, patent relationships. Apache-licensed software can be used in your commercial project for free. The only restriction is trademark use:

- don't name your project like it is an endorsement from Apache
- don't use their famous feather logo anywhere in your project or documentation.

Examples of software covered: Kubernetes, Swift, Firebase

BSD — Berkeley Software Distribution — is a permissive license that initially consisted of 4 chapters, but later the chapter, that demanded to inclose the copyright notice into all advertising materials, was excluded.

The BSD is often mentioned interchangeably with MIT due to very similar language and terms that accomplish largely identical goals.

The BSD License allows releasing proprietary software and incorporating this software into proprietary products as long as the original copyright and license requirement is fulfilled.

Examples of software covered: Django, React, Flutter (check out our article about Flutter)

MIT is a mythical license, as there is a myth that there is a license called MIT. This is because Massachusetts Institute of Technology initially used a lot of different licenses, like the Expat and the X11.

This license is permissive without copyleft and, as we already mentioned, is similar to BSD. It permits reuse within proprietary software with including a copy of the MIT License terms and the copyright notice. The mall size is an advantage of this license. The disadvantage is an inability to regulate patent relationships.

Examples of software covered: Angular.js, jQuery, .Net Core, Laravel

GNU GPL — General Public License — is the most known among open source licenses. Not because it is the easiest to understand, but mostly due to multiple mentions and references. As they call it “open license,” many think that the code delivered under GPL may be used in any possible way and programs made with it are entirely free of charge. However, statements are not valid.

The GPL is a copyleft license, meaning that any derivative works must be open-source and distributed under the same license, which makes it inappropriate to use in the proprietary software. As we already mentioned above, this is the most significant distinction to permissive free software licenses like BSD and MIT, widely-used less-restrictive examples. There are also GPLv2 vs. GPLv3.

Examples of software covered: Joomla, Notepad++, MySQL

GNU LGPL — the “Lesser GPL” — is a copyleft license, very similar to the GPL, from which it is derived, but with weaker copyleft requirements. It allows software being applied in a proprietary project under certain circumstances and to some parts of the project’s code. The LGPL is usually considered as a compromise between the strong copyleft licenses like GPL and permissive licenses like the BSD or MIT.

Examples of software covered: Qt, SharpDevelop

GNU AGPLv3 — Affero General Public License Version 3 — is also a copyleft license. Its terms are almost identical to terms of GPL with the additional paragraph, which allows users, interacting with the licensed program online to get the source code of the program. Thus usually it is not recommended to apply GNU AGPL for any commercial projects performing online.

Examples of software covered: SugarCRM, Launchpad

4. Licenses compatibility

In case of collective, combined or aggregated work the concept of licenses compatibility arises.

License compatibility is a characteristic of a license according to which the code distributed under this license may be integrated into a bigger software that will be distributed under another license.

For example, copyleft license's demand that any derived work combined from code under various licenses is applied to the copyleft license.

The widely-used licenses mentioned above tend to be compatible, but with some nuances, which could make it sometimes legally impossible to mix them. Thus we recommend investigating closer and be aware of license compatibility, before adopting them.

If you're a software developer, you're probably using open source components and libraries to build software. You know those components are governed by different open source licenses, but do you know all the license details?

In particular, do you know the sometimes-convoluted licensing conditions that could pose compliance challenges for your organization? Even one noncompliant license in your software can result in legal action, time-consuming remediation efforts, and delays in getting your product to market—all reasons for why ensuring open source license compliance is important.

- Identify all third-party components. Conduct a thorough inventory of all third-party software components used in your software, including both open source and commercial software.
- Developers should be wary of using AI-assisted coding tools, which may produce code with the potential for license violations and intellectual

property infringement. We are at the very beginning of practical, wide-spread AI-assisted coding. However, it's an open question—one that will likely have to be settled by the courts and government regulation—on the legal and ethical ramifications of AI-produced code.

- Assess license terms. Evaluate the license terms and conditions of each component and assess whether they are compatible with the intended use of the product.
- Check for compatibility between licenses of different components, as some licenses may not be compatible with each other.
- Use automated scanning tools to identify and track license obligations and restrictions for each component.
- Implement a compliance process for ongoing license compliance, including regular license scans and periodic reviews of license compliance procedures.
- Implement a review process and workflow for new or unfamiliar licenses.
- Ensure effective communication between legal, technical, and business stakeholders to properly prioritize and execute license clearance efforts.
- Document all license clearance activities, including license assessments and compliance procedures, to ensure a record of compliance efforts and to facilitate future audits.

The bottom line is that many open source licenses are open to interpretation based on the usage of the software. It can be very difficult to determine the legal risks of using open source software—especially for developers, who are not usually legal experts.

Open source licenses by risk

We've categorized the most popular open source licenses of 2022-23 into three broad groups, based on their popularity, terms and conditions, and their potential legal risks. This list of the 20 most popular open source licenses used by developers during 2022-23 also includes their risk classifications and whether the license has been approved by the Open Source Initiative (OSI). The OSI license review process ensures that software and licenses labeled as open source conform to community standards, and helps minimize license duplication and proliferation.

Data for the list was taken from the "Open Source Security and Risk Analysis" (OSSRA) report, an annual compilation of audits of commercial software conducted by the Black Duck® Audit Services team. Black Duck audits over the

years consistently indicate that the 20 most popular licenses will be found in approximately 98% of the open source in use.

The risk classifications are only a guideline and should not be used to make decisions about using the open source software governed by each license. Developers should consult their corporate policies and/or legal teams for guidance regarding license compliance.

Low risk: Permissive licenses

Permissive licenses generally do not have real limiting conditions. Rather, they usually require that you keep the copyright notice in place when you distribute your own software. This means you can use and change the open source software as needed as long as you keep the copyright notices intact. MIT and Apache licenses, the two most popular licenses currently in use, are in this category. We rate permissive licenses as LOW-risk licenses.

Medium risk: Semipermissive licenses

Semipermissive licenses usually require you to make any modifications to the source code available under the terms of the given license. Some of these licenses explicitly define what a modification is. For instance, a license might cite copying unmodified open source code into proprietary code as a modification. To comply with the license obligations, you would have to release the source code (original, modified, and newly added). Popular open source licenses in this category include the Mozilla and the Eclipse public licenses. We rate semipermissive licenses as MEDIUM-risk licenses.

High risk: Restrictive licenses

Some popular open source licenses, such as the GNU General Public License v2.0 or later and GNU Lesser General Public License v3.0 or later, are quite restrictive. Depending on how you integrate open source software with your proprietary software, you may face significant risk. In the worst-case scenario, you may be required to release your proprietary software under the same license—royalty-free. We rate restrictive licenses as HIGH-risk licenses.

On the MIT and CC licenses

Unsurprisingly, given its #1 position, the MIT license was found in 70% (some 721,000) of the open source audited for the 2023 OSSRA. You can be nearly certain that if your developers use open source or if you include third-party components in your software, you'll likely find the MIT license. As a permissive license that permits reuse within proprietary software, the MIT license has high compatibility and low risk with other software licenses.

On the other hand, Creative Commons (CC) licenses, while popular in the open source community, are specifically *not* recommended by their creators for software or hardware use. As the CC FAQ notes, "unlike software-specific licenses, CC licenses do not contain specific terms about the distribution of source code, which is often important to ensuring the free reuse and modifiability of software."

Many developers do use CC licenses for documentation, but some also inadvertently or deliberately attempt to apply a CC license to their code, resulting in Creative Commons Attribution ShareAlike 3.0 being the top license with conflicts in 2023 OSSRA audit results.

Managing open source license risk in the software supply chain

Whether you build packaged, embedded, or commercial SaaS software, open source license compliance should be a key concern for your organization. You need to determine the license types and terms for the open source components you use and ensure that they are compatible with the packaging and distribution of your software. Even companies whose software is not a commercial product and only used internally are still subject to the license terms of the open source components used in their software.

Begin by using an automated software composition analysis tool to create an up-to-date, accurate software Bill of Materials (SBOM) of all open source components used in your software, the versions in use, and their associated licenses. Compile the license texts associated with those components so that you can flag any components not compatible with your software's distribution and license requirements, or not compatible with licenses that may be used by other components in your software. It is important to ensure that the obligations of all licenses have been met, as even the most permissive open source licenses still contain an obligation for attribution.

Black Duck software composition analysis (SCA) enables development, security, and compliance teams to manage the risks that come from the use of open source. Black Duck's multifactor open source detection and KnowledgeBase™ of over 6 million components can provide an accurate SBOM, including licensing information, for any application or

container. And although the majority of open source components use one of the 20 most popular licenses, Black Duck provides an extra layer of information with data on over 2,500 other open source licenses that could potentially impose restrictions on the software your team writes. Tracking and managing open source with Black Duck helps you avoid license issues that can result in costly litigation or compromise your valuable intellectual property.

If your business is planning or Involved with an M&A

If your company plans to be involved with a merger and acquisition (M&A) transaction at some point, either as seller or buyer, you will want to involve your organization's general counsel or seek outside legal advice, as understanding licensing terms and conditions and identifying conflicts among various licenses can be challenging. It's vital to get this right the first time—especially if you build packaged or embedded software—because license terms are often more explicit for shipped software and harder to mitigate after the fact.

Knowing what open source code is in a company's codebase is crucial for properly managing its use and reuse, ensuring compliance with software licenses, and staying on top of patching vulnerabilities—all essential steps in reducing business risk. From an M&A perspective, a code audit enables a buyer to understand risks in the software that could affect the value of the intellectual property and the remediation required to address those risks. Sellers may employ an audit proactively to avoid surprises in due diligence, particularly given the amount of unknown open source in a typical company's code. An open source audit can be invaluable for companies wanting a better understanding of the code's composition. Using Black Duck SCA, expert auditors comprehensively identify the open source components in a codebase and flag legal compliance issues related to those components, prioritizing issues based on their severity.

The audit discovers known security vulnerabilities that affect the open source components, as well as information such as versions, duplications, and the state of a component's development activity. It also provides clues as to the sophistication of a target's software development processes. Open source is so ubiquitous today that if a company isn't managing that part of software development well, it raises questions as to how well it is managing other aspects.

If you're on the buy side of a tech M&A transaction, an open source audit should be part of the software due diligence process. Acquirers need to

identify problematic open source in the target's code before the transaction terms are set, and a trusted third-party audit is the best way to get a deep, comprehensive view. Sellers should prepare for questions about the composition of their code and how well they have managed open source security and license risk. Proactive sellers can prepare for an acquisition by having their software audited in advance.

By identifying open source code and third-party components and licenses, an open source audit can alert your firm to potential legal and security issues in an M&A transaction. With an open source audit, you can

- Avoid surprises
- Mitigate legal exposure
- Understand risks that may affect software asset values
- Resolve potential issues before they affect the transaction
- Build appropriate protections into the deal terms
- Plan integration and remediation of seller/buyer code

Significant monetary and brand risk can be buried in the open source components of acquired code. Evaluating that risk as part of an acquirer's due diligence must be a factor in the decision-making process of an M&A.

Proprietary software is any software with a license that restricts how it can be used, modified, or shared. Video games are a common example of proprietary software. If you purchase a video game (whether as a cartridge, disc, or digital download), you aren't allowed to make a copy of that game to share with friends or sell for profit. It's also likely you aren't permitted to modify the game's code to run it on a different platform than the one you originally bought it for.

Software users are typically held to certain restrictions with an *end-user license agreement* (EULA). If you've ever purchased software, you may have assumed you own that piece of software. However, if you've purchased proprietary software, it will likely come with a EULA that specifies you do not own the software. Instead, you're the owner of a software license that permits you to use that software. EULAs may also define how you can use the license itself, and they typically limit you from sharing it with others without the permission of the software owner (the software's developer or publisher).

Another legal instrument similar to a EULA is a *Terms of Service agreement* (ToS). Sometimes known as *Terms of Use* or *Terms and Conditions*, a ToS outlines the rules a user must follow in order to be allowed to use a program or service. It's more common to see an EULA included with software

that requires a one-time purchase, while ToS agreements are more common for subscription services and websites. Oftentimes, the first time you start a given piece of proprietary software, a dialog box will appear which explains the EULA or ToS and contains an **I Agree** button (or something similar) which you must click before you can use the program.

Software with such restrictions hasn't always been the norm. Before the 1970s, software was typically distributed along with its source code, meaning users were free to modify and share the software as they desired. With time, though, software publishers began imposing restrictions on these activities, typically with the goal of increasing profits by reducing the number of people who used their software but didn't pay for it.

This development had repercussions in the form of two closely related movements: the free software and the open-source software movements. Although the two are distinct, the free software and open-source software movements both argue that software users should be allowed to access a program's source code, modify it as they see fit, and share it as often and with whomever they like.

Conclusion

Surprisingly, while the majority of commercial licenses aimed to bound the user contain precise instructions, the most open-source software licenses, intended to guarantee freedom of usage, are written in an extremely complicated manner. We hope that our overview will help you to see the big picture on the licensing topic and will serve you as a crib. Here are some useful links for further research.