Free and Open Source Software: History, Philosophy, and Impact

JAMES CRAVEN, Winthrop University, USA

ABSTRACT

In the modern age of software development, it is easier than ever to share and consume software written by anyone, anywhere in the world thanks to the internet, and hosting services such as GitHub, GitLab, et cetera. In this day and age it is more important than ever to be educated about the ins and outs of open source software: its benefits, its dangers, and the precise terms of its licensing. This paper intends to mitigate these concerns by providing an overview of the history, philosophy, reality, and legality of FOSS development. An in-depth discussion of the various licenses available along with their terms of usage, role in the history of software development, and the reality of how these projects are managed will provide readers with all that they need to safely benefit from and contribute to the Free and Open Source Software Community, both in private, public, and proprietary settings.

1 Introduction

1.1 Precise Definition of FOSS

Free and open source software (FOSS) is a class of software that is distributed in such a way that the source code of the software is wholly available to the public (open source) and for which it is permissible to share, modify, distribute, or sell freely. Strictly speaking, for software to meet the definition of FOSS, it must follow four tenants of freedom. Users must be allowed to do the following things with the code without restriction: run, study, modify, or redistribute it [7]. Notably, this definition makes no mention of monetisation. It is a common misconception that the "Free" in FOSS means means free of charge, due to the ambiguous nature of the word free. The "Free" in FOSS means free in the sense of personal freedoms. Fortunato and Galassi (2021) reference several common colloquialisms

Author's Contact Information: James Craven, cravenj6@winthrop.edu, Winthrop University, Rock Hill, South Carolina, USA.

that—sometimes humorously—reference this confusion, such as "think 'free speech', not 'free beer'" [7].

1.2 The Importance of FOSS

There are many free and open source software projects that perform a critical role, both in the user and development space. Among the most famous examples include the the Linux kernel and the GNU (pronounced *guh-NEW*, /gəˈn(j)uː/) system utilities, which collectively make up the GNU/Linux operating system (typically referred to as just 'Linux'). As of May 2024, 96.3% of the top 1 million web servers use Linux as their operating system, approximately 85% of all smartphones use Android, a Linux distribution, and Linux's share of the desktop market usage is estimated to be increasing by 800-900 thousand users annually [5]. The vast majority of programming languages and/or their implementations are free and open source software. A few of the many examples are C++ (GCC), Python (cpython), Rust, and Go. Many languages are privately owned by companies initially before open sourcing, and others have both open and closed source implementations, such as Java.

Another important class of FOSS are user-space apps that cover a vast array of applications. For productivity there are entire office suites like LibreOffice, which is a drop-in replacement for the more well-known, proprietary Microsoft Office Suite. Image editors like Photoshop have FOSS counterparts in Gimp. Bourne Shell and its successors such as Bash and Zsh power not only Linux command lines, but the latter is also the default shell for MacOS, the second most popular operating system in the world [5]. FOSS is so widespread and capable that there exists fully functioning end-user operating systems that have options to completely disallow the usage of proprietary software, such as Fedora Linux. The importance of FOSS applications in all areas of software, from systems development, to web development, to producitivity tools cannot be understated. Without the existence of FOSS, it is highly improbable that any significant proprietary software could exist, be it due to a tool it relies on, a library in its codebase, or even the programming language that it is written in.

This paper will begin with a overview of the history of open source software, and its development from a hobbyist's pastime to its current status as a movement and a practice. From there, we will move into a discussion of the philosophy of the FOSS movement,

as well as divisions and the ever-present freedom-versus-practicality debate. After that overview, we will analyse how FOSS is carried out in practice, discussing first licensing schemes, then overall models for managing FOSS projects. A brief discussion of the importance of this topic will conclude the paper.

2 The History of Free, Then Open Source, Software

2.1 The Early Days, and UNIX

As discussed in Fortunato and Galassi (2021) [7], general consensus on the genesis of the Free and Open Source Software Movement is that it began with the "hacker" culture of the 1970s. In the early days of computing, software was tied to the hardware with which it came, due to a lack of widespread architecture standards. Software was not beholden to intellectual property rights, as it was tightly "bundled" with the computer itself, and hobbyists frequently modified programs and shared their modifications. This was beneficial for both consumers and corporations, as the more amateur developers contributed to their systems, the more support it would receive and thus the more marketable their systems would become. In 1969, an event known as the "unbundling" happened, which marked the beginning of the distribution of non-system-specific (or "unbundled") software.

Fortunato and Galassi [7] continue, explaining that anti-trust lawsuits levied at International Business Machines Corporation (IBM), the substantial time investment that software development required, as well as the rise of high-level languages such as C were the catalyst for this shift. The development of C coincided and eventually predicated the development of UNIX, an operating system being developed at Bell Labs, a subsidiary of the American Telephone and Telegraph Company (AT&T). Initially, UNIX was only made available for internal use at Bell Labs, however from the beginning of the early 1970s Bell Labs began licensing it out to other groups, primarily universities and other academic institutions. A select few corporations were also permitted access to the operating system. The company was unable to release only the UNIX program or market it as a stand-alone product, so they were forced to license it in this way, with the source code included. Due to its eventual complete port to C from its original assembly code and the high-level nature of C, it was the first operating system to be considered truly portable in the sense of being able to run on multiple devices. Many people used UNIX in this

time, and frequently interacted with developers at Bell Labs to bring improvements and updates to the operating system, which created a positive feedback loop in its adoption. Additionally, many variations were created, such as the Berkley Software Distribution Group's extended UNIX, which was released under a permissive license that became the foundation of the modern day BSD license. By the late 1970s UNIX had "thousands of users" [7].

2.2 The Birth of the Free Software Movement

The history outlined in Fortunato and Galassi [7] then goes on to explain that this uptick in usage prompted a regression of licensing for UNIX, as AT&T sought to maintain economic and intellectual property rights to UNIX, and limit its distribution. Around this time a programmer for the Artificial Intelligence Lab at the Massachusetts Institute of Technology (MIT), named Richard Stallman, became frustrated with the situation and began what he called GNU, an acronym standing for "GNU's Not UNIX." His goal was to create a UNIX- like operating system that was free for everyone. Stallman began advertising it as "Free UNIX," and thus the Free Software Movement was born. GNU would be a full operating system, including a kernel, something that other free UNIX variants at the time notably did not have. A kernel is the part of the operating system that allocates resources, such as memory, and provides access to the hardware to programs in the user, or non-kernel space. Additionally, GNU was to have a full suite of user utilities for all general purpose needs. From the beginning, the movement was equally ideological as it was technical, intending to provide all the software that a computer would need, entirely free. He wrote the GNU Manifesto, and he began recruiting for "[c]ontributions of time, money, programs[,] and equipment." From this the Free Software Foundation was born, and Stallman promised permanently free software by excluding GNU software from the public domain.

2.3 Moving on From UNIX

Concerns over the licensing issues of UNIX slowly lead users to begin veering away from the operating system. After extensive legal battles, BSD Group was able to keep its UNIX variant, and its free licensing allowed for proprietary distribution to become popular. From this operating system and its derivatives came the modern OSX (MacOS) and iOS

operating systems. The sharing of forks and other proprietary OS like early MS-DOS effectively wiped out UNIX. In the 1990s, GNU had completed its set of utilities, but had still yet to complete its kernel. Around this time then-Helenski-University-student Linus Torvalds began working on a kernel as a hobby project, and the Linux Kernel was born. It was eventually licensed under the GNU Public License, and merged with the GNU core-utils, GCC, et cetera to become GNU/Linux (hereafter referred to as simply Linux), the most popular open source operating system in the world [5]. The introduction of Linux to the public sphere drew more people into the Free Software Movement and it became central to the movements public image [7].

2.4 Open Source

Fortunato and Galassi's 2021 summary concludes as follows: the Free Software Movement, despite garnering much attention, had issues with practical corporate adoption. Larger corporate entities were hesitant to interact with free software due to the aforementioned ambiguity in its meaning. In the late 90s, the alternative term "Open Source" was derived, and from it eventually FOSS arose as a pormanteau of the two adjectives. Open Source itself was trademarked as a certification with a strict definition, and is currently managed by the Open Source Initiative (OSI).

3 FilOSSophy: Freedom vs Practicality

As mentioned in the previous section, FOSS began as "free software," under the supervision of the Free Software Foundation. It was not until much later that the rebrand to open source—or even FOSS—was set in motion. The difference between the two is more than a simple rebrand; as they represent two different approaches ideologically. This section delves deeper into the philosophy of these two movements: their core beliefs, key differences, and the debates that have occurred and continue occurring the FOSS community from both sides.

3.1 Free Software

The aforementioned creation of the the Free Software Foundation was formally outlined in a collection of documents, not least of which is known as the GNU Manifesto. In his manifesto, Stallman outlines his reasoning for creating free software, and in it outlines

the beginning of what would eventually come to be known as a copyleft license: "GNU is not in the public domain. Everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. [...] I want to make sure that all versions of GNU remain free" [22]. Stallman further outlined in other essays the precise definition of free software as he saw it, outlining the "Four Core Freedoms."

Numbered from 0–3 in reference to zero-indexed arrays in programming, Stallman issued his freedoms. Freedom zero states that a user must be able to run as they wish, for any purpose. Freedom one affirms the right of the user to study the functionality of the code underlying the program, and the ability to change the code as the user sees fits. He notes that a prerequisite to this right is that the code be available to the user. The third freedom, Freedom two, asserts that the user must be able to redistribute the source code, to "help [one's] neighbor." The final freedom is the ability to redistribute *modified* versions of the source code to others so that everyone can benefit from the changes. He again notes the necessity of freely available source code for this to function [21].

For Stallman, all of these steps were strictly a matter of right versus wrong, even going so far as to call the matter a "stark moral choice" [21]. His precise reasons were various, but to him, proprietary software was a form of division, and he left his place of work, saying that he would not be happy to look back on his career if he spent years aiding and abetting these "walls." He also cites having been on the receiving end of proprietary software, writing in reference to a time while working at MIT. There, a printer he made use of was subject to a primary license which he said made it difficult to work with because he could not alter its functionality [21]. Ultimately, Stallman saw himself as the ideological leader of a movement, and his ideas indeed garnered attention and popularity from 1983–1996 [13].

3.2 Criticisms, and the Open Source Initiative

As mentioned in Section 2.4, corporate entities were very hesitant to interact with something advertised as "free," even with the common speech versus-beer-analogy [13]. Many in the Free Software community began to take issue with this for practical reasons. Desiring to separate themselves from more ideological ideas, Eric Raymond and Bruce Perens founded the Open Source Initiative (OSI) in 1998 [12].

The main ways in which the OSI distinguishes itself from the FSF is in its communication and education on the ideas surrounding FOSS development. It aims to market the utility and benefits of open source software as a means to increase adoption. As part of this effort, and during the launching of the Initiative, the Open Source Definition (OSD) was created [12]. The definition creates a legal definition for, and extends the definition of, the Four Core Freedoms. This definition is used as "certification mark" for other open source licenses [12]. The exact OSD contains ten requirements to qualify. The first three encompass the Four Core Freedoms. The fourth requirement allows for restriction on modification of the author's code if and only if they allow a method of applying a patch to the program, and thus modifying it. Requirements five and six disallow personal discrimination and discrimination against purpose of redistribution, respectively. The seventh requirement disallows removing rights of any kind from those who receive second- hand distributions of the software. That is, all people who receive the software must be subject to the same license. Finally, the last three requirements dictate that the license must not be unique to any technology or product, and that they not make requirements of the software that is distributed along side the author's [11].

3.3 The Modern Philosophy of FOSS

Stallman would reject the ideas and philosophy that the OSI foments, writing that "[he] prefer[s] the term 'free software' because, once you have heard that it refers to freedom rather than price, it calls to mind freedom. The word 'open' never refers to freedom" [21]. While this may have been true at the time of his writing, current trends have shown otherwise.

In the modern FOSS community, the philosophy and terminology of the OSI seems to have become the dominant ways of conceptualising and referring to FOSS projects. It is increasingly rare to see a phrase like "ProjectName, the free X." Ultimately, the term open source or free and open source is generally preferred. For instance, the first line of the README for Blender, one of the most used 3D editing program is as follows "Blender is the free and open source 3D creation suite" [4]. Additionally many libraries for languages include "open" in the name, which does not appear to be the case for the word "free." To name a few examples: openssl, a FOSS library for interacting with SSL certificates; openjdk, the FOSS implementation of the Java runtime; openSSH, the SSH daemon that

comes pre-installed on most operating systems, including Microsoft Windows; openBSD one of the modern implementations of Berkley Software Distribution's UNIX fork; et cetera. On the other hand few use free directly in the name, and most do so under the Spanish word for free as in freedom, libre. To name some: LibreOffice, the open office suite; LibreWolf, a very recent FireFox fork; and FreeBSD, another fork of the original BSD. Admittedly, without a statistical study of open source naming conventions, this section is conjectural on the author's part; however, the trend seems to lean far more heavily in favour the neutral, corporate language of the OSI, whereas Stallman's ideas and terms are currently considered more fringe.

4 Open Source Licenses

Having discussed the history and philosophical basis of free and open source software, we move into the concrete details of how it works as a development model, beginning with the various licenses that commonly make up the space. There are two primary types of open source licenses: permissive licenses and copyleft licenses. Permissive licenses do exactly as the name suggests, they license the software to the end user in a permissive way, maximising the users ability to receive the rights outlined in the Four Core Freedoms. A copyleft license takes it a step further, and seeks to restrict the rights of corporate entities. These are the licenses that force all redistributions of the software to remain open source [15]. This type of license is sometimes referred to as a "poison pill" license as it forces anyone who wishes to use the work to also take the free software "pill."

4.1 Specific Licenses

There are numerous licenses that fit the Open Source Definition. The following subset have been selected for review due to their popularity and/or historical importance. All of these licenses are provided as templates on GitHub when a license is created, this fact being an additional reason for their inclusion.

4.1.1 GNU Public License. The history of the movement surrounding this license has already been discussed at length, both in sections 2.2 and 3.1. As was also mentioned before, the GNU Public License (GPL) is a copyleft license. It was the first of its kind, and was specifically designed to prevent proprietary redistributions of free software. In

addition to the practical legal protections it confers, the license also comes with a note about the ideological intent behind the license, and it describes the core beliefs of the Free Software Movement. According to GitHub's usage statistics, the GPL v3 has consistently been the 3rd most used open source license in the past 4 years [10].

The genesis of the GPL v1 was the combination (and generalisation) of the license with several programs produced by the GNU Project, the first of which being GNU Emacs [23]. This first draft of the GPL had some wording which Stallman and others considered too unclear, so a second version was drafted to further clarify these points [23]. This version of the license remained in use from 1991–2007, when the third and current version of the license was released. This license was again a minimally changed version of the former, but this time it added additional protections against practices that were not explicitly prohibited in previous versions, such as tivoization. According to Stallman, tivoization is a process by which a manufacturer of computerised appliances allow you to modify the programs running on them, but shut down if modified software is detected, thus creating a *de facto* prevention of code modification.

The permissions provided by the license are as follows: commercial use, modification of the source code, distribution of the source code, patent rights, and private use. As with most licenses, it limits liability of the author and denies warranty to the end user. Use of code under the GPL v3 license has the following stipulations: a copy of the license and the copyright notice must be included with distributions of the program, modifications must be disclosed and documented, the source code must be disclosed, and any modifications must also be released under the same license [9].

4.1.2 Massachusetts Institute of Technology License. The MIT License, much like the GPL, started as a collection of different licenses as opposed to a singular document. According to Jerome Saltzer, a source from MIT, in late 1983, a senior researcher at MIT, named David Clark, wrote networking software that began to garner wider interest which brought up questions of how the program should be licensed to users outside of MIT. The group decided that their primary objective was to influence the way software is done, and believed that licensing fees would be detrimental to that goal. As such, the MIT Licenses was born [20]. The researchers' goal of influencing the software world was undoubtedly realised. A few years after its initial release, Xorg, the developers of the X Window System

for Linux, distributed their windowing system under a slightly modified MIT license. The window manager would go on to be the most popular window manager on Linux for many years, being used on millions of devices [20]. On GitHub, the license is one of the most popular and well-known open-source licenses, and many amateur programmers use it to license one-off projects. According to GitHub's usage statistics, the MIT License has been the number one most used license since at least 2020 [10].

Unlike the GPL, the MIT License is not a copyleft license, but a permissive license. The license allows for usage of the software for any reason, regardless of intent. This includes commercial purpose. Like other software licenses, it disclaims liability and warranty. The only stipulation of the MIT License is that distributions retain a copy of their license. The simplicity of the license and its language has been a key factor in its wide adoption, a shortened excerpt from which may be seen below [14]:

> Permission is hereby granted, free of charge, [...] to deal in the Software without restriction, [...] without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, [...] subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Excluding the last portion of the license, which includes the aforementioned liability and warranty disclaimers, the above is essentially the entirety of the license. It is short, clear, concise, and protects the developer from the primary dangers of releasing software publicly.

4.1.3 Apache License. In 1993, a group of developers began sharing patches to the NCSA Networking Daemon, creating the beginnings of what would become the Apache Server software. By 1999, the group incorporated, becoming what is now known as the Apache Software Foundation. When it came time to release the software, they wanted to create a license that was open source, but more permissive [8]. Like the MIT License, the Apache License is permissive, not copyleft. It has been a very influential license, as it has consistently been the second most used named license on GitHub [10].

The license itself affords the following permissions: commercial and private use, modification, distribution, and patent use. Like the other license, it has disclaimers for liability

and warranty. Unseen thus far, however, is an explicit disclaimer that the license does not grant trademark rights. It is worth noting that this is true of other licenses as well; however, the Apache License notes this explicitly. Like the MIT License, the Apache License stipulates that the license must be included with distributions of the license. It also requires that any modifications of any kind be documented prior to redistribution [1].

4.1.4 Berkley Software Distribution License. The history of the Berkley Software Distribution (BSD) License was already touched on briefly in Section 2.1. When the BSD Group began distributing their variants of UNIX, initially they could not do so in its entirety, as some of the code was still from Bell Labs, and thus proprietary after AT&T began clamping down on its distribution [7]. To combat this, the BSD Group distributed individual components of the system pending the rewrite, and the permissive BSD License was born.

The first official version of the license is known as the BSD 4-Clause license or the "old license". Alongside the standard disclaimers for liability and warranty, it has the following four requirements, which give it its name [16]:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
(2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
(3) All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the organization.
(4) Neither the name of the copyright holder nor the names the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

With time, this version of the license became less popular, in part due to some of the restrictions it imposes. Today the four-clause license is not directly available as a template on GitHub, but one can find instead the 3- and 2-clause licenses [10]. The more popular

of the two, which until 2022 ranked as the fourth most popular license and more recently as the fifth [10], is the three-clause variant.

The three clause is largely identical to the "old license," but it omits the third clause which makes reference to advertising materials [18]. The other popular variant of this license is the two-clause variant which additionally removes what was originally the fourth clause, leaving only restrictions on redistributions [17]. This final variant, sometimes referred to as the "simple" variant, has ranked 10–12 on GitHub from 2020–2024 [10].

4.1.5 The Unlicense. Another license that is more recent and fairly popular is the Unlicense. In 2010, Arto Bendiken wrote a fiery blog post denouncing copyleft licenses such as the GPL, as well as the Free Software Movement as a whole. He likened the use of such copyright notices as threatening to send "men with guns" after those who infringe on the license [3]. Citing these reasons, he wrote The Unlicense.

The Unlicense seeks to do exactly what it implies, by entirely removing all copyright claim to the work, effectively permitting anyone to do anything with it. The license file—which Bendiken recommends to be saved to an UNLICENSE file instead of the traditional LICENSE file—says the following [2]:

> This is free and unencumbered software released into the public domain.
>
> Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.
>
> In jurisdictions that recognize copyright laws, the author or authors of this software dedicate any and all copyright interest in the software to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this software under copyright law.

To better understand Bendiken's reason for this license, it is necessary to examine his conception of licenses. Up until this point, this paper has envisioned open source licenses as one of two camps: permissive or copyleft. To Bendiken, this is a false dichotomy. He argues that a third type of license, public domain licenses, should also be considered a

part of FOSS. He argues that even permissive license are too restrictive, and that requiring attribution is "holding a gun to [one's] head" [3].

The Unlicense, despite its relative recency, is still a popular license, perhaps in part due to its inclusion as an option in GitHub's licensing templates. At the height of The Unlicense's popularity, it was found at the eighth position in GitHub's usage statistics, more recently falling to ninth [10].

4.2 Statistical Limitations

If conclusions can be drawn from GitHub's statistics alone, it is clear that most developers seem to prefer a permissive license over a copyleft license like that of the Free Software Foundation. Nothing can be said concretely about the reasons *why* this is the case, however. The vast majority of the licenses covered here are permissive, and that in and of itself may contribute to the high "preference" for permissive licenses. Also, the sample in question considers only GitHub which, while being wildly popular, is not the only place in which FOSS may be distributed. These licenses, in addition to a few others, are the only ones available in GitHub's template selection, and thus are more likely to be selected within GitHub.

5 Governance of FOSS Projects

FOSS projects encompass a wide variety of use cases, audiences, and sizes. Given the wide spread of possibilities, there is no one-size-fits-all solution for governing these projects. Generally, there are two main types of participants that are present in all governance styles: a committer and contributor. Generally speaking, a contributor is anyone who contributes time, resources, or code to a FOSS project. A committer is a bit less strictly defined, but their role is typically characterised by direct access to the code base. The precise authority or lack thereof that a committer has depends on the government model of the project [19].

5.1 Models of Government

5.1.1 Benevolent Dictator for Life. One of the most well-known leadership systems for an open source project is the Benevolent Dictatorship. This style of government is defined by the existence of a Benevolent Dictator for Life (usually shortened to BDFL). A BDFL

is a single person—or rarely, persons—that has final say over any decisions related to a project. Despite the name, which is largely used with humorous intent rather than desire for true association with a real-life dictatorship, BDFLs tend to be rather permissive in overall project direction, allowing for day-to-day decisions to be made by committers rather than micro-managing every detail of the project [19].

The most famous project that is an example of this governance model is the Linux kernel, which is dictated by the eponymous Linus Torvalds. The structure of the project is such that the various components are generally overseen by certain committers—which are referred to as maintainers within the Linux project—and Linus only steps in for conflict resolution and other major decisions [19]. Not all BDFLs are actually for life, however. An example of this is Guido Van Rossum, the Dutch programmer who created the Python programming language. Unlike Torvalds, Rossum has abdicated his position as of 2018, and the Python project is no longer run under this model [6].

5.1.2 Meritocracy. Another style of governance is a meritocracy. Meritocracies are more egalitarian; all members are considered equals, but it is possible to gain more influence and general standing with the community and thus the project by proving one's "merit," that is, contributing time to the project. Overall, it is the community who makes decisions about what direction a project should take in a meritocracy, but committers who have attained their status by donating time have an easier time affecting change. As mention in Section 5, a committer is one who has direct access to the codebase. When a contributor wishes to add to the project, they submit their patches for consideration, and the committers deliberate over its addition. Conversely, when a committer wishes to make a change, it must be agreed upon by the community. Ultimately, the role of a committer in a meritocracy is one of necessity more so than status [19].

5.1.3 Delegated Governance. It is often the case that the founder of a project, when disinterested in becoming a BDFL but still wanting something more rigid than a meritocracy, will create an official administrative body to govern the projects direction before relinquishing their temporary dictatorship. In this style of government, a temporary committee is appointed by the founder of the project, and from there delegates are appointed through various means common to real life governments, such as elections or appointments through various outlets. The councils and committees that head delegated

FOSS projects are a distinct, third class of project member (though there may be overlap between the council and committers, et cetera) [19].

5.1.4 Dynamic Governance. The final model discussed in this section will be the dynamic governance model. The dynamic governance model is defined by a wide dispersion of control throughout the community surrounding the project. Multiple committee-like structures exist that handle different parts of the project, and there is no top-down hierarchy across the project as a whole [19]. While they may seem similar, this model is distinct from a meritocracy. Unlike a meritocracy, the groups that hold decision making power in a dynamic governance model is generally an organised group of people with specific, definable structure.

5.2 Logistics

The nature of FOSS makes it paramount that a clearly defined method of communicating about, contributing to, and reviewing patches exists and is accessible to all members of the project's community. Traditionally this has been done with mailing lists and other decentralised methods of similar style. Nowadays, many newer projects have gravitated to git-based hosting websites such as GitHub and GitLab. Often feature requests, bug reports, and other topics with relation to the direction of these projects are hosted on these websites. While the day-to-day operations such as merging patch or pull requests are handled on sites like GitHub, often news, blogs, and documentation are hosted on associated websites as well. These tools are essential to the governance and operation of open source projects.

6 Conclusion

Since knowledge and experience are paramount to being a desirable candidate in the software development field, the ability to easily showcase this is crucial. One of the best ways to do this is by starting or contributing to projects. Open source is often an accessible way of building this portfolio.

As such, it is essential that junior software developers are familiar with the ins and outs of open source software development, as well as the applicability and ethics of the most common licenses that are encountered in the FOSS ecosystem. This paper serves to

provide critical background knowledge, as well as practical information on these licenses, open source software projects, and the challenges that lie within.

The usefulness of these concepts are not confined to the FOSS world, however. When making proprietary business software, it is often emphasised that programmers should not re-invent the wheel. In the age of the internet, it is easy to find software to serve just about any purpose. It is another question entirely to find software that can *legally* fit one's need in a proprietary setting.

Software engineers of today and tomorrow must be intentional and considerate with respect to licensing, whether it be licensing of the projects that they create, sub-licensing a library that their code base relies on, or carefully ensuring that the legal requirements of redistributed software are met. By familiarising ourselves with these concepts, we make better, more user-friendly, and more ethical software.

References

[1] Apache Software Foundation. 2004. Apache License Version 2.0. https://www.apache.org/licenses/LICENSE-2.0.

[2] Arto Bendiken. 2024. Unlicense Yourself: Set Your Code Free. https://unlicense.org/

[3] Arto Bendiken. 2025. Set Your Code Free. https://ar.to/2010/01/set-your-code-free

[4] blender. 2019. blender/README.md at blender/blender. https://github.com/blender/blender/blob/969e70fed8f1d0a07275bf2eec91c07a47b2a100/README.md

[5] Barry Elad. 2024. Linux Statistics 2024 - Market Share, Usage Data and Facts. https://www.enterpriseappstoday.com/stats/linux-statistics.html

[6] Carlie Fairchild. 2018. Guido van Rossum Stepping Down from Role as Python's Benevolent Dictator For Life. https://www.linuxjournal.com/content/guido-van-rossum-stepping-down-role-pythons-benevolent-dictator-life

[7] Laura Fortunato and Mark Galassi. 2021. The case for free and open source software in research and scholarship. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379, 2197 (2021), 20200079. doi:10.1098/rsta.2020.0079 arXiv:https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2020.0079

[8] Apache Software Foundation. 2021. ASF History. https://apache.org/history/

[9] Free Software Foundation. 2007. GNU General Public License Version 3. https://www.gnu.org/licenses/gpl-3.0.html.

[10] GitHub. 2023. https://innovationgraph.github.com/global-metrics/licenses

[11] Open Source Initiative. 2007. The Open Source Definition | Open Source Initiative. https://opensource.org/osd

[12] Open Source Initiative. 2011. History of the OSI | Open Source Initiative. https://opensource.org/history

[13] Mathias Klang. 2005. Free software and open source: The freedom debate and its consequences. *First Monday* 10, 3 (Mar 2005). https://firstmonday.org/ojs/index.php/fm/article/view/1211

[14] Massachusetts Institute of Technology. 1988. MIT License. https://opensource.org/licenses/MIT.

[15] Andrew Morin, Jennifer Urban, and Piotr Sliz. 2012. A Quick Guide to Software Licensing for the Scientist-Programmer. *PLOS Computational Biology* 8, 7 (07 2012), 1–7. doi:10.1371/journal.pcbi.1002598

[16] Regents of the University of California. 1990. BSD 4-Clause License (Original BSD License). https://spdx.org/licenses/BSD-4-Clause.html.

[17] Regents of the University of California. 1999. BSD 2-Clause License (Simplified BSD License). https://opensource.org/license/bsd-2-clause/.

[18] Regents of the University of California. 1999. BSD 3-Clause License (Modified BSD License). https://opensource.org/license/bsd-3-clause/.

[19] Dalia Ritvo, Kira Hessekiel, and Christopher Bavitz. 2017. *Challenges & Opportunities Concerning Corporate Formation, Nonprofit Status, & Governance for Open Source Projects.* Research Publication 2017-3. Berkman Klein Center for Internet & Society, Harvard University. https://ssrn.com/abstract=2937956 Harvard Public Law Working Paper No. 17-31.

[20] Jerome H. Saltzer. 2020. The Origin of the "MIT License". *IEEE Annals of the History of Computing* 42, 4 (2020), 94–98. doi:10.1109/MAHC.2020.3020234

[21] Richard Stallman and Mass Free Software Foundation (Cambridge. 2002. *Free software, free society : selected essays of Richard M. Stallman.* Free Software Foundation, Boston, Ma. https://archive.org/details/free-software-free-society-selected-essays-of-richard-m.-stallman-3rd-edition

[22] Richard M. Stallman. 1985. The GNU Manifesto. https://www.gnu.org/gnu/manifesto.html Originally published in Dr. Dobb's Journal, March 1985.

[23] Li-Cheng Tai. 2025. https://www.free-soft.org/gpl_history/