



In partnership with:



Releasing Internal Code into a New Open Source Project: A Guide for Stakeholders

November 2022

Ibrahim Haddad, Ph.D.
Executive Director, *LF AI & Data and PyTorch Foundation*

Contents

Infographic	4
Abstract	5
Introduction	6
Initial investigations	8
Make the business case to open source	8
Evaluate possible ways to open source	8
Project funding	9
Legal considerations	9
Confirm ownership of all the code.....	9
Conduct intellectual property review	10
Choose the open source license	10
Apply license terms to the code	11
Code clean-up.....	11
Project branding	12
Develop a trademark strategy and policy	12
Domain names	12
Creative assets.....	12
Register external accounts	12
Develop a certification/compliance strategy.....	12
Recruit business partners	12
Establish project governance	13
Set up project infrastructure.....	14
Apply recommended practices for your GitHub repo	14

- Project launch 15**
 - Prepare the announcement..... 15
 - Press and analyst relations 15
 - Announce and launch the project 15

- Summary of recommended practices for running an open source project 16**
 - License 16
 - Governance..... 16
 - Access..... 16
 - Processes..... 16
 - Development 16
 - Community 16
 - Community structure..... 16
 - Releases..... 17
 - Communication tools 17
 - Transparency..... 17
 - Documentation..... 17

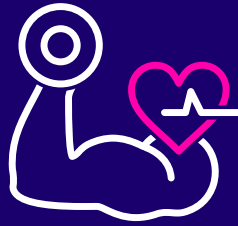
- Ongoing support 18**
 - Support the community..... 18
 - Support project infrastructure..... 18

- Endnotes 18**

License your project under an **OSI-APPROVED OPEN SOURCE LICENSE** to freely create and distribute derivative works.



Set up a transparent and equalizing governance model to **SUPPORT THE HEALTH, LONGEVITY, AND GROWTH** of the project.



Allow **OPEN ACCESS TO PROJECT RESOURCES** for anyone interested in participating in and contributing to the project.



DOCUMENT ALL PROJECT PROCESSES to maintain standards around commits, requests, peer reviews, and member roles, and be open to revising them with community feedback.



Manage the **DEVELOPMENT OF THE PROJECT THROUGH CAPABLE INDIVIDUALS MEETING QUALITY STANDARDS** at multiple levels of review before entering the final release.



Establish a community culture that strives for **ACCESSIBILITY, VISIBILITY, SELF-ORGANIZATION AND RESILIENCE.**



Structure the community to promote scalable activity from those who **ADD VALUE, WHETHER THEY ARE NEWCOMERS, ESTABLISHED MAINTAINERS, END-USERS, OR DEVELOPERS.**



Provide **STABLE RELEASES AT A DEFINED AND TRANSPARENT CADENCE**

that promote new functionality while maintaining reliability and security for users and developers.



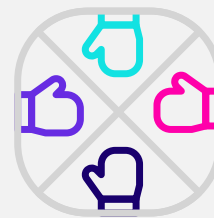
Establish **OPEN AND ACCESSIBLE COMMUNICATION TOOLS** to anyone wishing to participate in the project.



PROVIDE REGULAR COMMUNITY SUPPORT by holding meetings and events, issuing project updates, answering questions, pulling patch submissions, and creating accountable and trackable KPIs and goals.



To attract participation, **MAINTAIN TRANSPARENCY** in contributions, peer reviews, discussions, and maintainer or committer promotions.



MAKE AVAILABLE ALL DOCUMENTATION on architecture, APIs, tutorials, and guides for installation, development, and participation.

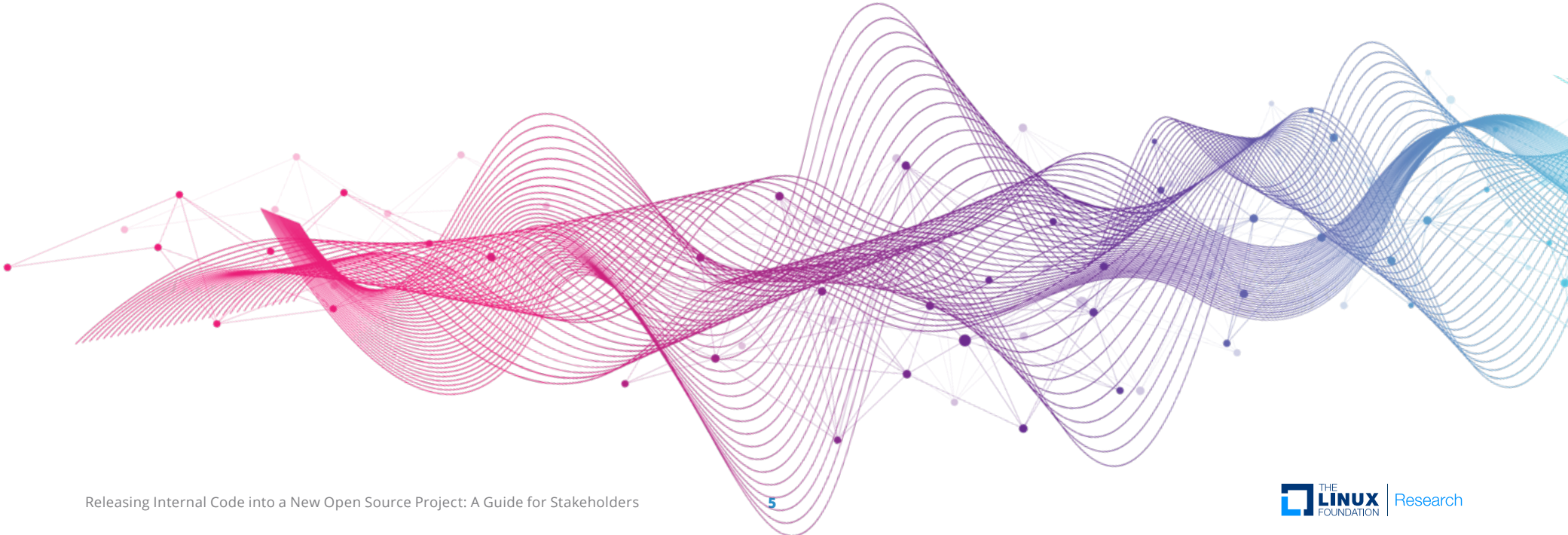


Abstract

Corporate participation in open source has reached an all-time high and continues to grow as companies realize the value of consuming and contributing to open source projects. The nature of corporate participation continues to evolve, as companies increasingly discover that open sourcing proprietary technologies can create new sources of value and stronger product ecosystems.

Open sourcing a proprietary technology involves far more than just making the source code available. There are many ways of building or joining communities to use and help maintain the project, which is why it should be a well-ordered and deliberate process.

For companies that plan to open source proprietary code as a standalone open source project, this paper offers a high-level overview of the process and provides a sample checklist that can help ensure that all tasks are properly captured and executed.



Introduction

Open source software (OSS) has been shifting the software industry into a new paradigm, moving from developing proprietary code behind closed doors to developing code that parties can share, modify, and redistribute openly. The key benefits of this shift include reducing development costs and software component complexity, developing reusable, common, off-the-shelf software assets, increasing flexibility, and benefiting from the innovation multiplier factor of community-driven development projects. Organizations that embrace the open source model as a positive means of building software will increase their chances of retaining their competitive advantage. Figure 1 illustrates the

various strategic advantages that OSS offers to organizations adopting it and contributing to it.

During the previous two decades, organizations have realized the benefits of using and contributing to open source projects in their products and services. This has created a trend of organizations setting up Open Source Program Offices (OSPOs) to manage all aspects of OSS, including the use of and compliance with OSS licenses, contribution to OSS projects, and community-building around key OSS technologies.

FIGURE 1
Why Open Source?

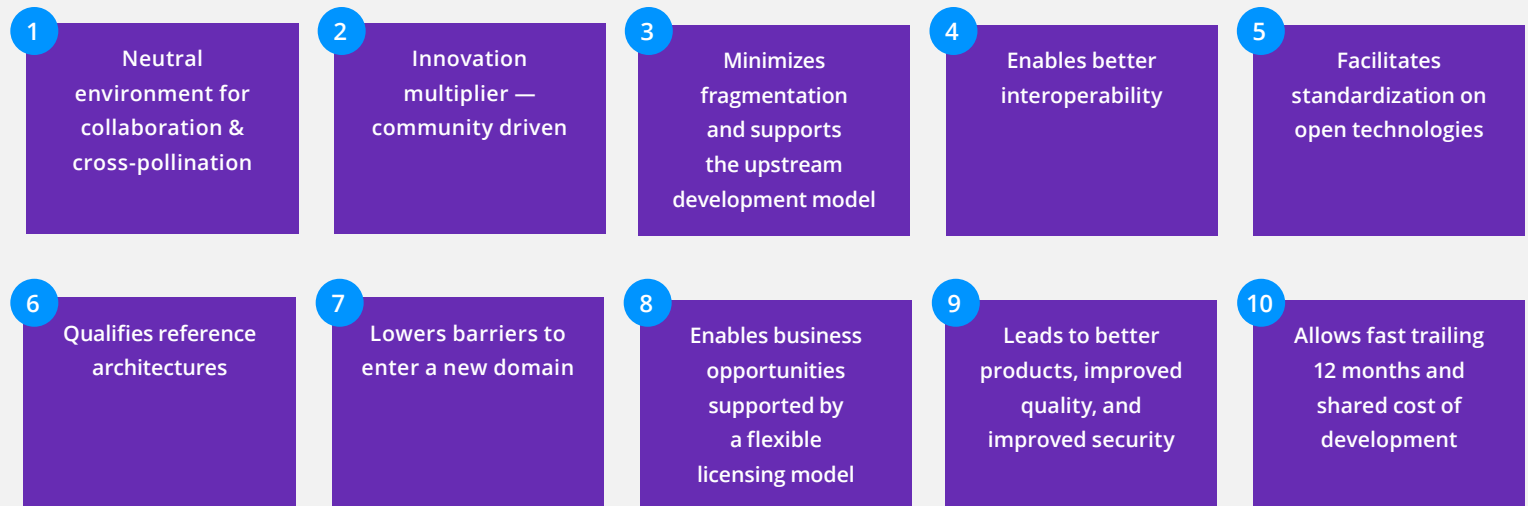


FIGURE 2

Enterprise open source ladder

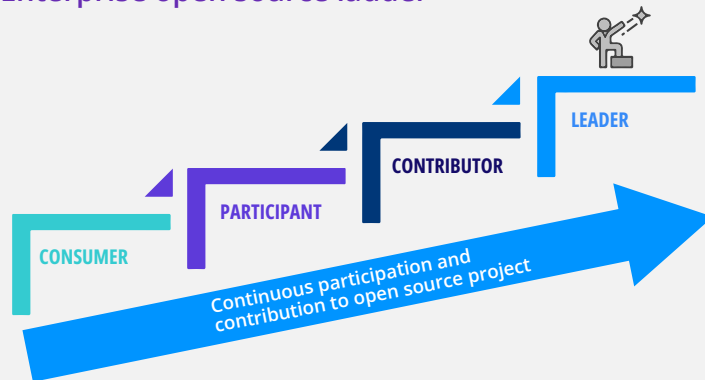


Figure 2 illustrates four primary OSS enterprise strategies: consumption of OSS, participation, contribution, and leadership. Each strategy requires an enterprise to succeed at the previous strategy, and how far an organization advances depends entirely on the enterprise. Engineering drives the early strategies of consumption and participation. Engineers use various open source components for their technical merits to speed up development, but they participate little in the projects that maintain these components. Over time, higher levels of the organization learn about the value of this OSS usage. As OSS gains traction, business needs begin to drive such OSS involvement, and OSS efforts contribute to a determined business strategy. Some companies achieve their goals as consumers. Other companies see strategic advantages in other stages of involvement and in most cases they set up an OSPO to oversee strategic planning and execution through these stages.

As part of the third stage—contributing to open source—organizations often choose to contribute key proprietary technologies to open source with various motivations, such as the following:

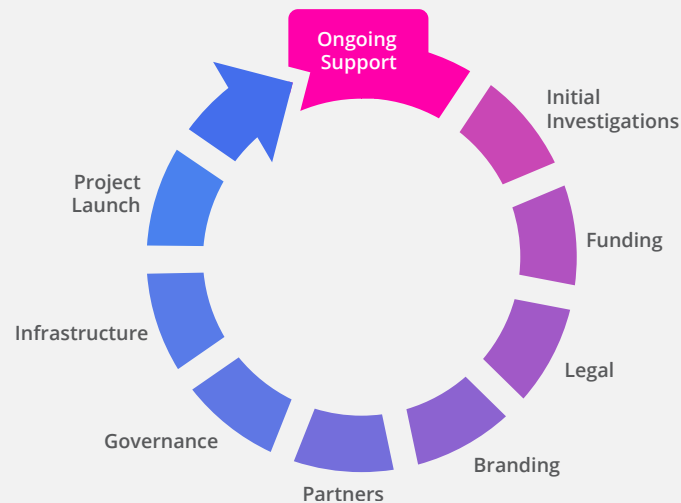
- Providing a reference implementation to a standard
- Ensuring that critical software remains viable
- Undercutting the competition
- Commoditizing a market
- Partnering with others and promoting goodwill in the developer community
- Driving market demand by building an ecosystem
- Offering customers the ability to support themselves and add custom features

Open sourcing with the wrong motivation will often have a negative effect on achieving the desired outcome and can disrupt the relation of the enterprise with the communities of specific open source projects.

This paper identifies questions to ask, practices to consider, and steps to take when making a proprietary technology open source. Figure 3 illustrates the various steps involved in the process of open sourcing internal code and launching it as an open source project. These steps are not necessarily executed in a linear order and several of them can be taking place in parallel. Our goal with this paper is to provide a basic template that organizations can adjust to accommodate their own policies and strategies.

FIGURE 3

Steps involved in the process of open-sourcing



Initial investigations

When open sourcing proprietary technology, it is important to thoroughly evaluate the reasons for the transition and align internal incentives and metrics accordingly. Open sourcing for the wrong reasons could have the opposite effect than is originally intended. To successfully open source a project, you must have the right reasons or motivations.

Make the business case to open source

There are many sound business reasons for open sourcing proprietary code, such as the following:

- Strengthening the ecosystem for the product or service you are building
- Improving product quality by engaging business partners and customers in enhancing features and fixing bugs
- Providing a reference implementation to a standard, thereby driving the adoption of your software as a de facto implementation
- Commoditizing non-strategic layers of a software stack
- Pushing the value line higher and forcing more innovation
- Partnering with open source communities and increasing goodwill within the developer communities

Equally, there are many counterproductive reasons to open source proprietary code. These arguments should act as red flags:

- You want others to maintain a codebase you still need so that you can stop investing in that code
- You want to retire code with unique functionality that you will no longer maintain or use in your products

- The source code links directly to code you cannot release under an open source license

Now that you have a business case for open sourcing your code at your organization, the next step is to determine the actual path to open source.

Evaluate possible ways to open source

There is no single way to achieve the possible goal, and it's not an exercise that your organization has to do alone. In most cases, there are multiple options that you can explore, such as the following:

- Evaluate the technology and determine whether you should open source any other components
- Analyze existing open source projects your company could join as a major participant, reducing your need to create new infrastructure and a new community
- Explore the possibility of launching the planned open source project with some of your existing clients and partners
- Evaluate the option of launching and hosting the planned open source project in an open source foundation with a record of launching and sustaining successful open source projects.

Project funding

Once you have made the business case for open source, you'll need a project plan and time-phased budget covering the costs to launch and maintain the project over time. Some of the costs are one-time and others are recurring. Examples of such costs include the following:

- Internal legal efforts leading to posting the code publicly
- Ongoing IT project infrastructure and cloud credits (when applicable)

- Trademark management
- Creative and branding (logo, website, signage at events, etc.)
- Project management
- Regularly scheduled open source license compliance and security scans
- Community events and hackathons

Legal considerations

On the legal side, there are five major activities that need to be executed. These include:

1. Confirming ownership of all the source code intended for open source
2. Conducting intellectual property review
3. Choosing an open source license
4. Applying the license terms to the code and updating the license information in the source code
5. Cleaning up the source code before posting it in public

In the following subsections, we cover these various activities and provide recommendations where applicable.

Confirm ownership of all the code

One of the risks in open sourcing proprietary technology is the accidental inclusion of third-party proprietary code as part of the open sourced code. Before releasing any code under an open source license, it is highly recommended that an organization holds all the rights and permissions necessary to open source the code.

Some of the steps in this exercise include the following:

- Auditing the source code with a software composition analysis (SCA) tool¹
- Identifying third-party source code—open source or commercial
- Determining whether you have the right to open source any found third-party commercial code under an open source license; if the answer is no, you cannot open source some third-party code, then you need to provide alternate code

Conduct intellectual property review

Software likely subject to patent or other intellectual property claims is not an ideal candidate for open source release. Here are questions to ask to help you with this exercise:

- Does the code disclose or realize any inventions the company plans to protect through patents?
 - If the answer is yes, then you need to decide whether to remove the code, establish an IP policy, or make a nonassertion pledge. In all cases, your legal counsel will make the appropriate recommendations for next steps when such a scenario arises.
- Will the source code release trigger patent claims against the open-source software?
 - If the answer is yes, then you have to remove and replace the protected code or seek appropriate licenses or permissions.
- Does the name you chose for the open source project (assuming you're starting a new project) protectable under trademark law? Does the name or any trademarks associated with or registered to the project present any risk of infringement claims?

Choose the open source license

The license of an open source project determines the rights to use, copy, modify, and distribute the code. The choice of license for an open source project is an essential factor in determining the openness of the project. Open source projects should only use licenses that the [Open Source Initiative](#) has approved. Such licenses allow software to be freely used, modified, and shared. To be approved by the Open Source Initiative, a license must go

through its [license review process](#) to confirm that the license satisfies its [Open Source Definition](#) (OSD). You may come across many other licenses that are incompatible with the OSD. Most of these licenses are “Source Available” licenses that commonly include restrictions or limitations on the use and/or distribution of the software. These restrictions often render the licenses incompatible with the OSD.

It is always recommended to adopt an OSI-approved open source license. The choice of the specific license depends on the specific goals you want to achieve as an organization. We preview in this section some of the questions that will drive a discussion on this topic and help you make a decision on the license to adopt.

- Do you want to relinquish control over how others use and distribute the code?
- Do you want to allow others to use the code in commercial programs and products?
- If others use the code in their program and sell it for money, do you want a percentage of those revenues?
- If others use and distribute the code and improve it (e.g., fix bugs or add features), do you want them to contribute those improvements back to the project?
What licenses will you accept for contributed code?
- Will you require a Developer Certificate of Origin (DCO) or a Contributor License Agreement (CLA) as a contribution requirement?

There may be additional questions to consider as part of this exercise, which is mainly driven by your legal counsel and technology leaders within your organization.

DCO

The DCO sign-off process ensures that every single line of code accepted into a project has a clear audit trail. It is a developer's certification that they have the right to submit code for inclusion into the project. The Linux kernel process, for instance, requires all contributors to sign off their code, which indicates that the contributor certifies the code, as outlined in the [DCO](#). The signature communicates that the contributor has created or received the contribution under an appropriate open source license that incorporates it into the project's codebase under the license indicated in the file. The DCO establishes a chain of people who take responsibility for the licensing and provenance of contributions to the project.

CLA

Some projects require either developers or their employers to sign a CLA. Unlike the DCO, the text of CLAs can vary significantly from project to project, so the terms of any given CLA may have different effects. The purpose of a CLA is to ensure that the guardian of a project's outputs has the necessary ownership or grants of rights over all contributions to allow them to distribute under the chosen license. In some cases, this even means that the contributor will grant an irrevocable license, which allows the project to distribute the contribution as part of the project.

Apply license terms to the code

Once the source code has been cleaned up following the recommendations provided in a previous step, it is time to apply the license terms to the code. This exercise includes the following steps:

- Add a license file in the root directory of the component containing the full license text. For instance, if you are posting the code on GitHub, you will provide a `LICENSE.md` file containing the full text of the open source license

- Add license header and copyright notice to every source code file
- Clearly designate the license on the project's website, any frequently asked questions (FAQ) that you provide, and the download page if applicable
- Use [SPDX² License List](#) "short identifiers" in source files

Code clean-up

Another risk to open sourcing a project is the inclusion of private information, confidential communications, and trade secrets. To minimize this risk, you can take the following actions:

- Remove any exposure of nonpublic application programming interfaces (APIs).
- Remove any comments containing employee names or personally identifying information, product code names, road maps, future product descriptions, or disparagements.
- Remove any unused or obsolete code from the source code to increase the likelihood of the community's making contributions.
- Create and include a file that contains the license and copyright notices of all third-party software and, if applicable, make the source code available.
- If the source code has dependencies on third-party code, then provide the necessary information to the community; it's preferred that the code does not have any dependencies on non-open source components.
- Remove any third-party proprietary code.

Project branding

Project branding includes a number of activities that should be considered and are discussed in the following subsections.

Develop a trademark strategy and policy

- Agree on a name/mark for the project.
- Perform a knockout search to determine whether the registrations will succeed.
- Specify internal contact for trademark (if not counsel).
- Develop a registration strategy:
 - Which classes of goods/services apply?
 - Which countries to prioritize?
- Register the tradename and trademark.

Domain names

- Register domains and set up redirects.

Creative assets

- Create the logo, logo package, and visual assets.
- Create and publish on the website the logo usage guidelines.

Register external accounts

- Set up the project organization name on GitHub.
- Set up @projectname on various social media platforms such as Twitter, LinkedIn and Facebook.

Develop a certification/compliance strategy

- Decide the criteria projects must meet to claim compatibility with the parent project.

- Create a specification document and tools that can verify whether a custom build of the project complies with the specification.
- Agree on the name/mark for the certification program.
- Develop a trademark policy/FAQ if you wish to control the use of the project name. Ask these questions to drive a conversation on the topic:
 - May distributors, user groups, or developers register domain names that include the project's name?
 - Will the project run certification programs allowing others to use the mark for modified products?
- Create a certification test suite.
- Establish a contract with a testing facility.
- Schedule the first year of plugfests.

Recruit business partners

- Approach business partners that will benefit the most from the project for public support on launch day.
- Secure commitments from key partners to encourage employee participation in the project and have some basic commitments.
- Approach compatible projects, communicate how the new project will benefit them and prepare them for the announcement.
- Anticipate conflicts where existing projects misinterpret the launch as competition and defuse them before they start.
- Give business partners early access to project source code.
- Work with partners to establish a joint value proposition and reference stack for shared customers.

Establish project governance

Governance determines who has influence and control over the project beyond what is legally required in the open source license. A project's governance model establishes a collaboration framework that addresses difficult questions, such as the following:

- **Contributions**
 - Who makes decisions for code inclusion and releases, and how?
 - Who can be the lead maintainer or architect for the project (larger projects have more than one)?
 - How can the project contributors become maintainers or committers?
- **Direction and Finance**
 - How can the project raise money, and who decides how this money is spent?
 - Should the project have a Technical Steering Committee (TSC) or a Conformance and Certification Committee? Who can be on them?
 - Who decides the project's direction and road map?
- **Transparency**
 - Who can participate in the discussions and decide on critical matters?
 - How transparent are the decision-making processes?
 - Can anyone follow the discussions and meetings that take place in the project?
- **Reuse**
 - What compliance requirements are there for redistributing, modifying, or using the software?
 - How can the project enable contributors and downstream redistributors to comply with these requirements?
- **Copyright and Trademark**
 - Who owns the copyright on contributed code?
 - How can users license the project's branding?

Typically, the initial maintainers of the project form the TSC of the project. These individuals are likely from the founding organization(s) of the project. The goal is to grow the TSC over time to include high-value contributors.

- **Project governance**
 - Identify members of the TSC.
 - Identify primary duties of the TSC, such as the following:
 - » Overseeing software architecture and implementation activities
 - » Drafting the release plan and road map
 - » Working with other open source projects on which the new project depends
 - » Setting the criteria for accepted/rejected code
 - » Managing source code security issues
- **Project processes:** A project with a high degree of openness will have clearly defined processes for how things work in the community and how to contribute to the project. For starters, a clear development process should outline how to incorporate code into the project, the release process and schedule of the project, and any requirements developers need to meet to get their code accepted. This should also include guidelines for participation that demonstrate community best practices for things like patch submissions, feature requests, bug reports, and signing off on code contributions.
 - Feature request
 - Release management
 - Code submission
 - Bug reporting
- **Project agreements**
 - Develop a third-party contribution agreement to govern how the project will manage contributions from the community.

Set up project infrastructure

- Documentation
- Project website
- Wiki
- Community communication channels
 - Mailing list
 - Live chat (e.g., Slack, Internet Relay Chat (IRC))
- Collaboration platforms
 - Wiki
 - GitHub repositories (or manage your own git servers)
- Bug tracking and feature requests
- Build system

Apply recommended practices for your GitHub repo

1. Use the [REPOLINTER](#) tool created by the [TODO Group](#) to identify common issues in GitHub repos.
2. Secure your GitHub account with [two-factor authentication](#).
3. Ensure that every repo includes a `LICENSE` file.
4. Add a `README` file to your repos welcoming new community members to the project and explaining why the project is useful and how to get started.
5. Add a `CONTRIBUTING` file to your repos explaining how to contribute to the project to other developers and your community of users. At a high level, the file would explain what types of contributions are needed and how the process works.
6. Add a `CODEOWNERS` file to define individuals or teams responsible for code in a repository.
7. Add a `CODE_OF_CONDUCT` file that sets the ground rules for participants' behavior and helps facilitate a friendly, welcoming environment. While not every project has a `CODE_OF_CONDUCT` file, its presence signals that this is a

welcoming project to contribute to and defines standards for engaging with the project's community.

8. Provide documentation on the release methodology, cadence, criteria, etc.
9. Document your project governance and make it available on the project's repo.
10. Add a `SUPPORT` file to let users and developers know how to get help with your project. You can either add how and where this file handles security issues, put it at the project's top-level `README`, or refer to security documentation.
11. Set up an issue template and pull request templates that help you customize and standardize the information you'd like contributors to include when they open issues and pull requests in your repository.
12. Achieve and maintain your project's [OpenSSF Best Practices Badge](#) (previously called the Core Infrastructure Initiative Best Practices Badge).
13. Identify who will handle security issues (this could be a team) and set up a separate email account.
14. Consider having the project become a CNA (CVE Numbering Authority).
15. Include an SPDX short-form identifier in a comment at the top of each file in the repo wherever reasonably possible.
16. Adopt the [GitHub DCO app](#) to enforce a "Signed off-by:" tag in each commit. The DCO is an easy way for contributors to certify that they wrote or otherwise have the right to submit the code they are contributing to the project. The app enforces the DCO on Pull Requests. It requires all commit messages to contain the `Signed-off-by` line with an email address that matches the commit author.
17. Use English as the default universal language for anything you publish on GitHub. You can support a second language, but English should be the primary language of communication toward a universal audience.

Project launch

Prepare the announcement

- Brief launch partners.
- Check that all project infrastructure is running, secure, and scalable.
- Subscribe key project personnel to project mailing lists.
- Make sure internal developers join and continually monitor the live chat.

Press and analyst relations

- Establish launch strategy and timeline.
- Draft press release and get signoff from all involved parties.
- Identify spokesperson and media contact.

- Create internal and external FAQ.
- Manage ongoing press and analyst relations.
- Develop the ongoing public relations/analyst relations strategy.
- Engage a PR/AR firm if needed to deliver fully on the strategy.

Announce and launch the project

- Release source code.
- Publish a road map, even if it is preliminary.
- Follow the open source development model.
- Monitor effects of PR/AR strategy across touchpoints.

Summary of recommended practices for running an open source project

License

OSI-approved open source license offering the freedom to create and distribute derivatives.

Governance

A governance model that gives equal footing to all current and future contributors to the project. Open source projects with an open and transparent governance model have better chances to grow, have a healthy environment, and attract developers and adoptees.

Access

Project resources are accessible to any users or developers interested in the project. Anyone can participate in the project, and any participant can earn committer rights by contributing and building trust with the project's community.

Processes

- General project processes are documented for requesting a feature, reporting bugs, submitting code, etc.
- Source code contributions are only committed through the project's defined process for incoming contributions.
- All code goes through a peer review process.
- The process to become a committer/maintainer/reviewer is enforced by the project for consistency.
- The project's community revises its processes based on incoming feedback to ensure they continue to meet the project's needs as it grows and scales.

Development

- Responsibility for development is allocated to the individuals with the best capacity to deliver.
- The project enforces quality standards when merging code.
- The project implements multiple levels of review before entering the final release.
- Peer review is mandatory and public.

Community

- Accessible to newcomers—open development generally strives for inclusiveness.
- Focused on visibility with emphasis on open decision-making processes and communication.
- Self-organizing; individuals contribute in their areas of interest or those of their employers.
- Resilient to organizational change, given that leadership comes with experience. If individuals cease to participate, there are others to take their place.

Community structure

- Meritocracy drives advancement and acceptance. Contributors who provide the most value to the community are granted project leadership roles.
- The project welcomes newcomers who have the freedom and access to participate in public discussions, development, and testing.

- The project's hierarchy is scalable because it consists of maintainers who oversee specific bodies of code in levels that can be added or removed as needed based on the size of the community.
- Anyone can submit patches, and both developers and users are involved in the testing process. The roles of developer and user are closely integrated with open source development, allowing users to have a more direct path to influencing the project.

Releases

- To protect certain users from the instability of rapidly developing software, projects provide stable releases that restrict the addition of experimental features to provide a reliable version that better supports use cases that rely on stability.
- Weekly or monthly stable releases provide users and developers with the newest functionality after it has been tested.
- Long-term stable versions extend to longer periods and often only include security patches and bug fixes.
- The project has a defined cadence for its releases, with set goals per release.
- The release cadence and the goals to be met by each release are known to all project stakeholders.

Communication tools

Tools, including mailing lists, Slack, and IRC, are available and open to anyone wishing to participate in the project.

Transparency

Open source communities must be as transparent as possible to attract new participation, such as contribution transparency, peer review transparency, transparency of discussions, and transparency of promotion to committer or maintainer.

Documentation

Availability of documentation covering architecture, APIs, installation guides, developer guides, development processes, participation guides, tutorials, etc.

Ongoing support

After the project has launched, it is essential to monitor the vitality of the external community and support the project in various areas to nurture it and support the growth of its community.

Support the community

- Meet regularly with key stakeholders.
- Issue regular project updates through the website, PR, and social media.
- Respond to questions from the community in communication channels.
- Review patch submissions and pull them into the codebase as necessary.
- Coordinate events to cultivate community and promote the technology.
- Develop a set of KPIs for project success, track these metrics, and develop and implement plans to ensure the attainment of these goals.

Support project infrastructure

- Keep content on the website and wiki up to date.
- Provide ongoing guidance to trademark counsel.
- Manage trademark over time.
- Manage domain registrations and renewals.
- Monitor and moderate communication channels (mailing lists, IRC, forums, etc.).
- Maintain press and analyst relations.
- Develop the ongoing PR/AR strategy, and engage with a firm to provide an appropriate level of service.

Endnotes

- 1 SCA tools are applications that support software development teams to ensure open source license compliance and improve the security of the code. At a high level, they perform automated scans on source codebases. The tools also help the team identify open source components and their license and flag any known security vulnerabilities.
- 2 The Software Package Data Exchange® (SPDX®) is an open standard for communicating software bill of material information between organizations as well as from upstream open source projects into an organization.

Conclusion

There are many ways to successfully open source proprietary technology. This paper provides a high-level overview of the process and can be used as a base for a more detailed internal plan. It is important to acknowledge that this checklist may not be complete and differs between organizations and projects. The goal is to provide the most common tasks associated with open sourcing internal projects and make them available to ease the process. While this process may seem complex and lengthy, many organizations have successfully followed similar procedures to bring internal code to market as an open source project and, in the process, have automated a lot of tasks and used project management tools to coordinate and track all tasks.

For more information on creating successful open source projects and working with open source communities, please visit the Linux Foundation website for a host of free resources available to help you with your open source journey.

Acknowledgments

The author would like to express his sincere appreciation to his Linux Foundation colleagues Hilary Carter, Jason H. Perlow, and Melissa Schmidt for their valuable reviews and feedback. This report has benefited immensely from their experiences and contributions.

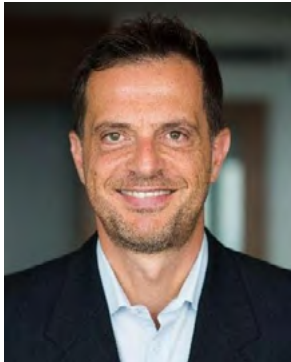
Linux Foundation resources

- E-book: [A Deep Dive into Open Source Program Offices](#)
- E-book: [Guide to Enterprise Open Source](#)
- E-book: [Open Source Compliance in the Enterprise](#)
- E-book: [Open Source Audits in Merger and Acquisition Transactions](#)
- [Linux Foundation Enterprise Guides](#)
- [Linux Foundation Open Compliance Program](#)—Resources to support organizations with open source compliance.
- [TODO Group](#)—Open community of practitioners and organizations that collaborate on best practices, tools, and other ways to run successful open source programs.
- [Software Package Data Exchange®](#) (SPDX®)

Feedback

The author apologizes in advance for any spelling mistakes or possible errors and is grateful to receive corrections and suggestions for improvements via ibrahimatlinux.com/contact.html

About the author



Dr. Ibrahim Haddad is Vice President of Strategic Programs at the Linux Foundation. He focuses on facilitating a vendor-neutral environment for advancing the open source AI platform and empowering generations of open source innovators by providing a neutral, trusted hub for developers to code, manage, and scale open source technology projects. Haddad leads the LF AI & Data Foundation and the PyTorch Foundation. His work and the work of both foundations support companies, developers, and the open source community in identifying and contributing to the technological projects that address industry and technology challenges for the benefit of all participants.

Twitter: [@IbrahimAtLinux](https://twitter.com/IbrahimAtLinux)


Website: ibrahimatlinux.com

Fun project: [Tux NFT Club](#)

Disclaimer

This report is provided “as is.” The Linux Foundation and its authors, contributors, and sponsors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to this report. In no event will The Linux Foundation and its authors, contributors, and sponsors be liable to any other party for lost profits or any form of indirect, special, incidental, or consequential damages of any character from any causes of action of any kind with respect to this report, whether based on breach of contract, tort (including negligence), or otherwise, and whether they have been advised of the possibility of such damage. Sponsorship of the creation of this report does not constitute an endorsement of its findings by any of its sponsors.

 twitter.com/linuxfoundation

 facebook.com/TheLinuxFoundation

 linkedin.com/company/the-linux-foundation

 youtube.com/user/TheLinuxFoundation

LF AI & DATA

Part of the Linux Foundation, LF AI & Data supports open source innovation in artificial intelligence, machine learning, deep learning, and data. LF AI & Data was established to support a sustainable open source AI ecosystem that makes it easy to create AI and Data products and services using open source technologies. We foster collaboration under a neutral environment with an open governance model to support the harmonization and acceleration of open source technical projects.



Founded in 2021, Linux Foundation Research explores the growing scale of open source collaboration, providing insight into emerging technology trends, best practices, and the global impact of open source projects. Through leveraging project databases and networks, and a commitment to best practices in quantitative and qualitative methodologies, Linux Foundation Research is creating the go-to library for open source insights for the benefit of organizations the world over.



Copyright © 2022 [The Linux Foundation](https://www.linuxfoundation.org/)

This report is licensed under the [Creative Commons Attribution-NoDerivatives 4.0 International Public License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

To reference the work, please cite as follows: Ibrahim Haddad, "Artificial Intelligence and Data in Open Source: Challenges and Opportunities for Mass Collaboration at Scale," foreword by Dr. Seth Dobrin, March, 2022.