O'REILLY®

Compliments of aws

Open Source in the Enterprise

Andy Oram & Zaheda Bhorat

opensource.amazon.com

Open Source in the Enterprise

Andy Oram and Zaheda Bhorat

Beijing · Boston · Farnham · Sebastopol · Tokyo

Open Source in the Enterprise
by Andy Oram and Zaheda Bhorat

Editor: Michele Cronin
Production Editor: Kristen Brown
Copyeditor: Octal Publishing Services, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery

July 2018: First Edition

Revision History for the First Edition
2018-06-18: First Release

978-1-492-04195-5

[LSI]

Table of Contents

Acknowledgments

Creating and growing open source projects and communities takes a village. Throughout this book, we reference projects, processes, books, reports, best practices, and all forms of contributions developed by foundations, companies, communities, and individuals. We trust that these will be incredibly valuable resources on your open source journey and will provide the additional depth needed on each topic.

On this, the twentieth anniversary of open source, we'd like to acknowledge our deepest thanks to every individual member of an open source community who has contributed to open source in any way for their significant and valuable contributions, in sharing code, tools, lessons, practices, processes, and advocacy for open source for the benefit of all.

We'd also like to thank our reviewers who made extensive comments and helpful suggestions—Cecilia Donnelly, Karl Fogel, James Vasile, Chris Aniszczyk, Deborah Nicholson, Shane Coughlan, Ricardo Sueiras, Henri Yandell, and Adrian Cockcroft. And finally, thanks to both the O'Reilly Media and AWS teams for supporting this book to bring these resources together.

—Andy and Zaheda

Open Source in the Enterprise

Free and open source software is everywhere, frequently taking over entire fields of computing. GNU/Linux is now the most common operating system, powering data centers and controlling Android devices around the world. Apache Hadoop and its follow-on open source technologies brought the big data revolution to a wide range of organizations, whereas Docker and Kubernetes underpin microservices-based cloud computing, and artificial intelligence (AI) has overwhelmingly become the province of open source technologies such as TensorFlow and Apache MXNet. The major players in computing—such as Amazon, Apple, Facebook, Google, Huawei, IBM, Intel, Microsoft, PayPal, Red Hat, and Twitter—have launched and maintain open source projects, and they're not doing it out of altruism. Every business and government involved with digital transformation or with building services in the cloud is consuming open source software because it's good for business and for their mission.

It is time for organizations of every size and in every field to include free and open source software in their strategies. This book summarizes decades of lessons from open source communities to present a contemporary view of the trend. We'll help you effectively use the software, contribute to it, and even launch an open source project of your own.

Not only do companies get better software by utilizing open source, but the dynamics of working in that community-based fashion opens up new channels for creativity and collaboration within these companies. Conversely, institutions that fail to engage with open source will fall behind those that use it effectively.

Finally, it's worth mentioning that trade secrets and confidential business plans can coexist with open source engagement. If even the US National Security Agency and UK Government Communications Headquarters can use open source software, you can, too.

Why Are Companies and Governments Turning to Open Source?

There are solid business reasons for using, supporting, and creating open source software. Benefits include the following:

Multiplying the company's investment
: Open source benefits from the famous principle: "The smartest people in every field are never in your own company." At best, an ecosystem of innovation will grow up around an open project. Evidence that opening a project pays off financially comes from a recent report prepared under World Bank auspices. Careful tracing of contributions to their project—a form of geospatial software called GeoNode—showed that the World Bank's subsidiary had invested about one million dollars in the project but had benefited from an estimated two million dollars invested by other organizations.

Benefiting from the most recent advances
: The AI projects mentioned in the introduction are a good example. Your data scientists will want implementations of the best and most up-to-date algorithms, and these implementations will usually be open source. There is no need to reinvent the wheel in-house. Furthermore, your company can innovate more quickly by using tools and existing code that you can get with a simple download and installation process.

Spreading knowledge of the software
: When the code is open—and especially when a robust community grows up around it—adoption is broader. This initially takes effort on the company's part, but it leads to more people throughout the industry understanding the code and the contribution process.

Increasing the developer base
: Broader adoption, along with wide discussion of the source code, translates into a larger pool of talented developers from which the company can hire to work on the code and related projects.

Upgrading internal developer skills
: The spread of knowledge goes in many directions. Developers already recognize that the best way to learn good coding skills is to work on an open source project because they can study the practices of the top coders in the field. These benefits spread to the company that employs the open source developers.

Building reputation
: Most people want to work for organizations they can boast about. Adopting open source—both the code and the practices that go with it—shows that

your organization is cool. And if you can release your own code as open source and win adoption for it, you prove that your organization is a leader in your field, adept at the best development practices.

Recruiting and retaining developers

Good developers want to work on exciting projects that affect large groups of people. They also want their skills and contributions to be widely recognized, and they enjoy the interactions they can have with peers around the world. All these considerations lead them to gravitate toward open source projects, and if your competitors are more successful than you in supporting such projects, developers will bring their talents and reputations to those companies instead of yours.

Faster startup of new companies and projects

In the frenetic pace of today's social and business environments, a startup or new division needs to go from concept to product in months, not years. Working with a community, both on existing software and on your own innovations, saves you time and lets you focus limited employee time on critical competitive parts of your product.

Many governments have launched major open source policies and initiatives. As France and the United States demonstrate, we are now seeing a shift from the use of open source to policies that encourage the development of open source and investment in open source communities. Some have committed to an "open source first" strategy, requiring vendors as well as internal developers to use open source licenses and practices wherever possible. For example, the government of France has stated that all agencies must do future code work in open source. With such policies, agencies can revise obsolete, expensive, slow procurement practices that have been notorious for causing failed software projects and outrageous cost overruns. For governments, open source becomes a staging ground for the latest, more responsive software practices that have proven more efficient and productive in other sectors.

Furthermore, governments are realizing that each agency's needs are similar to other agencies, around the nation and around the world. Open source means, at least, that the investment made by one agency can save money for all the rest—and, at best, that the agencies will share requirements and collaborate in the classic open source manner to create software that helps governments better serve their citizens everywhere. Open source collaboration also opens opportunities for smaller companies, citizen developers, and nonprofits to contribute to innovation in government services. Finally, the software creates a common standard that fosters interoperability for many kinds of development.

More Than a License or Even Code

In open source, a productive community and its accompanying practices are just as important as the code itself. Officially, of course, open source is defined by a license. Popular ones include the GNU General Public License, the Mozilla Public License, and the Apache License, all of which go through occasional version changes. But in practical terms, you need much more than a license to have a thriving open source project.

Many people cite the principle "community before code" from the Apache Software Foundation. At a conference, one open source community leader explained the principle as follows:

> If you have great code and a dysfunctional community, people will leave and the code will atrophy. If you have dysfunctional code but a great community, people will improve the code.

That observation extends to the culture of your own company, where it becomes crucial to create a community among developers from different teams and let them work productively in the larger project community.

We summarize these practices in this book, along with references to resources that will help you on your open source journey. Here are a few places to go for more information:

- An extensive reading list provided by the Linux Foundation. Perhaps the most cited books from this list are Karl Fogel's *Producing Open Source Software* (O'Reilly, 2018) and Eric Raymond's classic *The Cathedral and the Bazaar* (O'Reilly, 2009).
- A comprehensive set of guides from the Linux Foundation. These are developed by members of the TODO Group, a collaboration among open source program offices and contributors from companies that have adopted open source principles, practices, and tools.
- Resources and answers to questions from the open source community at Opensource.com.

So powerful are open source practices and community behavior that many companies mimic open source techniques internally, in a process called InnerSource. You can pursue this process, described in another O'Reilly Media report, in parallel with open source participation or on its own.

Most organizations—unless they grow organically out of a healthy open source project—greatly underestimate the role of open source culture. This culture is strikingly different from the secretive, hierarchical, management-driven cultures of most companies today. Values of open source projects include listening skills, transparency, collaboration, sharing expertise, mentoring, recognizing merit wherever people demonstrate it, respecting diversity of needs and opinions, and disciplining one's own ego to accept criticism.

Many companies establish an open source program office (OSPO), where open source is fostered, supported, nurtured, shared, and explained both inside and outside the company. OSPOs are vital for larger organizations that have invested heavily in consuming and contributing to open source software. OSPOs from different companies also collaborate to share best practices that sustain open source development and communities. You can learn more about OSPOs via case studies by the TODO Group.

Groundwork for Understanding Open Source

Before discussing open source software from three angles—how to adopt software developed elsewhere, how to contribute to a project, and how to launch a project of your own—let's quickly try to dispel a few myths:

Open source software is low quality or less secure
: Now that major companies are involved in open source, this myth is not cited so often, but it persists in attitudes that often go unstated. People accustomed to typical procurement processes have trouble believing that something distributed without cost can be high quality. In fact, open source projects have replaced the need to charge for licenses through a number of other funding strategies. But the key issue is that strong open source projects adopt strict quality processes, which your organization can also adopt for your own benefit. As for security, flaws occur in both open source and closed software. Neither is guaranteed to avoid breaches. Experience suggests that transparency and a large development community in open source lead to faster fixes and faster distribution of the fixed software.

Open source software lacks support
: Popular open source projects have many sources of technical support from both organizations and individuals. The open code is a great advantage because you are not locked into a single company for support. Smaller and younger projects might not have yet developed this ecosystem of support, so getting support here might require you to devote more developer time and draw on informal help from the community. Likely enough, you will stumble over a critical bug that requires immediate attention someday, and you will be thankful that you can apply your own developers or a hire a developer to fix the bug instead of waiting for an indifferent vendor's fix.

Open source software projects are unmanaged and chaotic and free for the taking
: As you will see through the course of this book, successful open source projects have well-defined processes for decision-making, review of code, and dealing with users like your organization. You must follow certain rules

when you use code developed by others. The code almost always has a license, but with different rules from proprietary code. If one of your developers copies code found on the internet into your own products, you will almost certainly be violating a license, and it's a bad practice for legal and other reasons. These points receive more discussion at the Open Source Initiative and Software Freedom Conservancy. Later sections of this book explain current practices for accepting open source code into your organization.

Using open source software requires you to open your code
: This is something of a reverse of the previous myth. Certainly, you need to be aware of what the license requires before you use open source software. Some licenses have rules for contributing back changes you make, and as you'll see later, you benefit by doing so. (These licenses are sometimes called "viral," but their users dislike the negative connotations of that word; "copyleft" is a more neutral term.) Most open source projects, even those with rules for contributing back code, are distributed as libraries that you can link into your own code without opening the functions you write yourself (for instance, the GNU Lesser General Public License).

You can gain a user base and community by releasing code on a public repository
: Opening code out of laziness never works. Open source projects do have an advantage over proprietary ones in gaining adoption, but only if you treat the project as a respected element of your business strategy. Open source projects realize value only as part of an active community. In many cases, the interactions inherent in that process are in and of themselves highly valuable to participants. But open source dynamics reward ongoing investment. You'll see more of how to do this during the course of the book. Inactive projects produce declining benefits over time as the costs of stagnation add up.

An open source project will occupy all your developers' time with support requests
: In open source, you trade the time you spend on support for the contributions you get back from the community. Certainly, you must budget time for support, but your company can control how much time to put in and can pull back in order to meet deadlines on internal projects or control excessive support costs. In a successful open source community, all members engage in education, and your company is not solely responsible for providing it.

Terminology: Free and Open Source

The terms *free* and *open source* appear interchangeably in this book, because with negligible exceptions, everything that falls under the definition of free software also falls under the definition of open source, and vice versa. The term *free software* is used by those who want to emphasize the aspects of liberty, privacy, and sharing, whereas *open source* is used by those who emphasize its practical and business benefits.

Do not use the obsolete term *freeware*, which used to refer to programs whose developers kept the source code closed while distributing the executable files cost free. This is not free software as currently understood. To be truly free (or open source), the source code must be available and must be under a license that allows its users to modify and redistribute it.

Adopting and Using Open Source Code

We trust you are curious what open source code can offer, and perhaps eager to find code that can solve a business need. This section summarizes the key processes you need to adopt to successfully use other people's open source code. The resources cited earlier in the book go into much more detail.

Create and Document an Internal Open Source Policy

Your development team should know exactly what open source code it is using, and where. This tracking is done by your OSPO or by a virtual team of employees if you have not yet set up an OSPO. Tracking has two main purposes: establishing an audit trail to demonstrate that you are using the code properly, and ensuring that you comply with the license obligations on your third-party open source dependencies. Collecting this information is critical for many reasons; most organizations do so through automated tools in their development cycle.

Writing a strategy paper is valuable to educate managers and employees. Think big and aim for the end state that you are trying to achieve. At the same time, frame the broad, high-level goals in the context of business outcomes. Here are some points that have been have used successfully to explain what open source can do for an organization:

- Attract and retain talent
- Increase agility, drive innovation, and accelerate the creation of business value
- Reduce costs and improve efficiency by focusing your staff on writing business logic and by eliminating reinvent-the-wheel heavy lifting

- Generate revenue or gain market share, either through your product or through thought leadership

Break down your strategy into milestones. This allows you to assign ownership and speed up the delivery of the multiple processes that are needed. In terms of strategy, think about these:

- Open source governance and policies that clarify to the broader company how and when it can use open source
- Policies specifying how developers can contribute to external open source projects: roles and time spent
- Encouraging an open policy in applicable software projects from the start among technology leadership and enterprise architecture groups
- Starting an InnerSource model in tandem, in which you adopt open source practices for internal development across your entire company

Because the adoption of open source crosses many organizational boundaries and can lead to new organizational structures, you might need to explore the creation of new policies that allow developers to collaborate internally across those structures.

It is important to find a senior sponsor who will help open doors and champion your cause. This can be the most difficult task we describe in this section, but it is critical. You will need the sponsor on your journey. Make your strategy paper compelling, and they will buy into your high-level, long-range vision. Aim for CTO or CIO support, but be prepared to have to work your way up to that.

Legal staff need to be trained to understand licenses that might be radically different from anything they have dealt with previously. The marketing and PR teams also need to discuss the effects of open source on your practices, quality, and responsiveness to customers. This requires them to study open source practices, collaborate on communication with the communities, and translate these new practices into interactions with your customers. Members of the open source community that you're joining (and probably hiring from) can explain to your marketing team the importance of attending and supporting events held by the community. In addition, the sales team needs to understand licensing well enough to answer questions about using and extending the code, when presenting your solution to customers.

If you don't have a clear process, you risk having someone incorporate open source code informally without following good practices. Not only could this violate the code's license, but it deprives you of the benefits of properly using open source code. For instance, critical bug fixes are released regularly by open source projects, and you need to know where you're using this code in order to install the fixes. Instituting clear processes also allows your employees to become valued members of the community and represent your organization's needs there. Finally, a clear and well-communicated policy leads to dramatically more participation and more awareness of your open source initiative throughout the company.

Formalize Your Strategy Through an OSPO

Developers can participate informally as members of open source communities, but companies who want to fully benefit need to centralize the logistics for supporting open source: legal vetting, project vetting, recruiting developers to work on the code, sponsoring projects and events, managing communications and community relations, and so on. Most companies entering open source have therefore created an OSPO to promote it and handle the associated tasks. Several OSPO leads have collaborated via the Linux Foundation's TODO group to assemble sample open source templates and policies that can be useful to companies getting started. Your OSPO can sponsor the activities we describe in subsequent sections.

Build Ties Throughout the Company

After creating a policy, make sure that all of your developers know it. Many other areas of the company, such as the legal and procurement teams, need to be on board as well.

First, create a supportive community of open source practitioners within the company. Developers and others who have worked in open source communities can do this at a grassroots level, with or without support and guidance from management. The advocates spend time evangelizing and educating other employees about open source through activities such as regular lunchtime sessions, webinars, and presentations at team meetings.

A training course given to all developers in the company will make sure everyone has the same understanding and expectations about using and contributing to open source.

These outreach activities all help demystify open source and accustom the employees to its culture. Most important, those activities uncover other potential champions, so you can begin working with them early on.

Assess Potential Projects

There are plenty of places to find open source code. GitHub and GitLab are two well-known code hosting sites (both use the popular Git version control software developed originally by Linus Torvalds). A search for keywords describing your need (for instance, "employee management") might well turn up thousands of projects. So be careful to determine that an interesting project really meets your needs. One Linux Foundation guide focuses on the process, and the book *Open Source for the Enterprise* by Dan Woods and Gautam Guliani (O'Reilly, 2015) also offers guidelines for judging project readiness. Following is a list of typical things to check for:

Code quality
: You can assess this by examining the code, checking the ratings (stars) if the project is on GitHub, looking at the number of reported bugs, and seeing what people say about the project online. All code has errors, and you want to see that people report them, so the presence of bug reports is a good thing —so long as the important bugs get fixed.

Active development
: Has the project put out any new releases recently, or at least bug fixes? Are there many pull requests, which indicate interest in the code?

Project maturity
: How long has the project been in existence? Is there an active community? Are there a number of people maintaining the code? Does it have funding? Are books, videos, and other forms of education available?

Level of support
: Can you get help to install and maintain the software? Is there good documentation for the project?

Health of the community
: Are the mailing lists active? Do developers respond quickly and positively when bugs are reported? Are people polite and productive when responding to issues? A growing community is a good sign but not a necessity, because a project that serves a small user base might turn out to be just right for you. On the other hand, an inactive or declining community is a warning sign.

Public decision making
: Are all choices debated on the public list? If you get a sense that leaders make important decisions privately and just report results to a mailing list or repository, it's a red flag indicating that you might not be able to influence the direction of the project. You might not think such influence is important now, but as you become dependent on the software, you might want to have a say in the future.

Governance/commit access
: Is there a documented way for new contributors to gain commit access? If you invested your time and expertise in contributing to the project, you will want it to support an increase in your responsibilities.

Security reporting
: Professional projects will provide a way for people who discover security flaws to contact the leadership privately so that the flaws can be fixed before advertising their existence.

Comply with the License

The OSPO or team responsible in your organization for tracking the use of open source code should also make sure you obey the license. This requires attention from both developers, who understand the technical implications of the license, and legal staff. Participants from the open source community have formed the OpenChain Project to make open source license compliance simpler and more consistent and thus encourage more companies to use open source software. Recognizing that it can be difficult to find the license or attribution of code, another open source project called ClearlyDefined helps users to find and maintain this compliance information. The book *Open Source Compliance in the Enterprise* by Ibrahim Haddad (The Linux Foundation, 2016) covers compliance in depth.

Build a strong relationship with your legal team and partner with them. They will almost certainly be dealing with incoming work around open source: for instance, they need to approve doing the work under an open source license and check the contributor agreements under which your developers submit code to the projects. This is an opportunity to collaborate on the unfamiliar aspects of open source, build a strong relationship, and earn trust. Use these relationships to help position open source policies in a more favorable light for development and developers.

Also form ties with your procurement colleagues. You can add value here by recommending specific additional items to put into procurement contracts around open source and help the team review contracts with third parties.

Manage Community Code as Seriously as the Code You Create

Even though an outside organization developed the open source code, you need to manage its use within your organization. It should be subject to testing and to checking for back doors and other flaws. You need to set goals and deadlines for incorporating the code into your own, just as when your employees write code for your company, and devote resources to making sure everything goes as planned.

Of course, some tasks are different when you get code from outside. You need to act as part of the community that maintains the code. For instance, you need to integrate your developers with the community's version control system; this might require an account where the community hosts the code, such as GitHub or GitLab. If you alter the code (discussed in more detail in "Contributing to

Open Source Projects" on page 17), you should set up a branch of the version control system for your version or use your own internal version control.

Change Your Reward and Management Structure

Developers working with an open source community will do things differently from closed projects and will need additional skills in communication and negotiation. Your reward structure must reflect the extra tasks you require of these developers. You must set aside developer time for such tasks as learning the tools used by the project, participating in chats and on mailing lists, mentoring less experienced colleagues, checking and modifying other peoples' contributed code, and attending conferences or leadership meetings for the open source project.

In particular, as developers and other employees spend time on efforts less directly related to the deliverables of their specific teams, you will need to adjust performance evaluation criteria for both them and their managers. This is especially true for companies that pay performance bonuses. It's crucial to align the goals and incentives within the company with success factors on the open source project. In turn, when employees attend and speak at conferences, blog about their work on the project, and participate in governance, their contributions reflect well on your company and allow you to reap later benefits such as hiring new developers.

Besides approving these changes, which can be extensive, to the compensation plan for many employees, managers must also recognize that software deadlines are dependent on activities by the community, not just internal employees. Deadlines for software projects are notoriously difficult to set accurately, even when they are run totally by internal teams, so the new dependencies on outsiders might simply help management recognize reality. If the community decides that a feature your company needs is low priority, go ahead and use your own resources to finish it in a timely manner—and contribute the improvements back to the community. Other companies will be doing similar things, and all will benefit.

Ego and Open Source

It is natural to take pride in creative output. Software is rarely any longer a single person's ingenious inspiration in the way Donald Knuth developed $\TeX$ and Metafont on his own. But even when software requires team effort, working tightly together as a team provides its own sense of a unique shared identity. Furthermore, the team can boost a developer's personal needs. It's easy in a close-knit group for each member to know the other's strengths. A manager can see exactly how someone contributed and base pay increases and promotions on those contributions.

This psychological and practical reward structure is profoundly changed by open source. We must consider how each developer finds personal fulfillment and an acknowledgment of their contributions in an open source context.

When starting or joining an open source project, a team must give up some control and must learn to work within much more amorphous groups of people who come and go, all motivated by different goals. But ultimately, the individual has an even greater chance to shine on an open source project, because every contribution is available for the world to see. It might turn up in distantly related projects that were never imagined. And demonstrating solid contributions that other people choose to adopt is a wonderful boost to a job application. Open source also rewards people who have skills in communication, project and people management, design, web development, documentation, translation, and much more in addition to coding chops. All contributions matter in a project.

This high visibility of open source projects can also scare managers who worry that their best contributors will be lured by higher salaries or more prestigious jobs offered by a competitor. The solution to that problem is to be that competitor. Make sure your employees have the time to contribute to the open source software you use, both as coders and in a leadership role. Give them the freedom to run with good ideas and provide them the resources to implement useful tools that developers or teams come up with. If you have star performers, encourage them to join boards and speak at conferences. They will help turn your company into a magnet for other top developers.

Participating in a Project's Community

The health and success of your company will now depend, to a greater or lesser degree, on the health and success of the open source project. Not only should you devote employee time to participation as community members (a topic covered in more detail in the section "Change Your Reward and Management Structure" on page 12), but your company can be a valuable resource. Your experiences with the software, including pull requests, bug fixes, and change requests, can help the community improve the code for everybody's benefit.

Participation means doing the same things other contributors do. Developers working with the code can do the following:

- Post questions, ideas, and bug reports
- Contribute fixes and new features, signing the contributor agreements that give the project rights to the contributed code
- Attend code-a-thons and conferences, always being transparent about whom they work for

- Become committers (advanced developers who are trusted to change the core code repository) and participate in the project's governance, as they grow into their community and leadership roles

Their participation will help them gain recognition from the community of users and developers for both their own work and your company's support.

Many successful projects in free and open source software follow community principles commonly known as the Apache Way. Although members of these projects might not use exactly the same terms as defined by leaders of the Apache Software Foundation, the principles are recognizable in the projects' day-to-day conduct.

Developers will come to your company with these values or will learn them as they work on open source projects. What's really difficult for the company is that similar values—tempered by the need for confidentiality in certain areas, which every organization experiences—must percolate from the developers through the entire organization. Advocates for open source work need to schedule discussions with management and get their buy-in on the culture needed to reap the benefits of open source. The value of transparency and community participation must be accepted at all levels of the company by managers, giving developers the freedom to determine where best to invest their efforts.

Here are some of the activities developers need to do when joining an open source community:

- Check the code of conduct and contributor agreements, which are important to adhere to
- Become adept at the tools used by the project
- Review the project's documentation (which varies in comprehensiveness and completeness), starting with the README file
- Become comfortable with the process for making contributions provided by the project
- Ask questions and become familiar with other contributors through participation in mailing lists or groups
- Start taking items to work on from the project's TODO list
- Submit bug fixes

After your employees gain an understanding of the open source code, your organization might find reasons to make your own changes. Often, a project will not have the exact features that you want. You might also decide to make different performance trade-offs. Open source projects encourage users to send all of their changes back "upstream" to the project repository. Depending on the license and how you plan to use the code, you might be required to do so. Well-designed projects are modularized so that people who want your changes can adopt them and others can ignore them.

It can take a long time—even years—to get your changes into the main project, or *core*. The project leaders might hold off for many reasons: they might decide that no one else needs your enhancements, or be afraid that they're buggy, or worry that they'll have a negative effect on performance. You also can run aground over cultural barriers: your developers simply might not understand the community well enough to gain its attention and trust. But it's worth addressing any concerns the project leaders have and persist in trying to work together with them, not on your own.

You might need to set up a separate branch for development or clone the code and take it internal to your organization. This is called a *fork*. There are usually ways to reintegrate your fork with the main branch later, if the core committers agree to that.

When you contribute or push your fixes back, you get the enormous benefit of a thorough and uncompromising code review. The code that ultimately goes in will end up much better than the code you submitted: fewer bugs, better performance, and an architecture that is more generalized for future uses.

Furthermore, if your code becomes part of the core, you can simply install any updates that come from the project, secure in the knowledge that they have your enhancements and will work with them. If you fork the code, you must either give up getting new versions (and suffer from any security flaws and bugs the original code contained) or tediously reapply your enhancements to the new version, testing carefully and checking for changes that invalidate your own work.

The cost of maintaining forks is quite high over time, and most companies that lack the training to contribute back to the original project also lack the internal processes that would allow them to benefit from the original project's continued development. They don't download and incorporate upstream bug fixes, security patches, and feature improvements because the effort of coordinating with the original project increases as the original and custom versions diverge.

All that said, you might find that the changes you make to code have no prospect of being accepted into the core. Or, you might have reasons for withholding your changes; for instance, if you need to bury secrets in the code for regulatory or competitive reasons. Some companies choose to maintain a separate branch, publicly or privately, and do the grunt work of reapplying enhancements to new versions. Sometimes, companies contribute part of their changes back but withhold others. Any withholding, however, can reduce the value and appeal of the project.

Quality and Security: A Comparison of Open and Closed Source

Just as you evaluate a vendor for quality, good business practices, customer support, and general reputation before buying its goods, you need to evaluate open source software. The section "Assess Potential Projects" on page 9 offers some pointers as a start. With open source software, the risks of making the wrong choice are that you don't get the quality you expected or that interest in the project dries up and contributors move elsewhere. If the latter problem happens, you probably should move too, although you have the option of taking responsibility for the code and recruiting others to keep it going.

In return, open source has several advantages over a closed-software vendor. Open source offers you other options if your vendor no longer suits your needs; that is, goes out of business, abandons the field you're in and takes the product in a different direction, ratchets up the cost of the product precipitously, refuses to fix a bug or add a feature that is important to you, or even inserts malware that tracks your activity. All of these things have happened in proprietary software. Open source software grants you control over your future direction. You can switch vendors and decide how to fix problems in ways that best suit your needs.

Much ink has been strewn about on claims that closed source is higher quality or lower quality or more secure or less secure than open source code. The question has not been resolved either way. To encourage robust development practices, the Core Infrastructure Initiative at the Linux Foundation has created a "best practices" guide covering a wide range of common issues, from source control to testing and code analysis.

Security experts almost universally agree that an open source process is better for secure code and standards, because experts everywhere can evaluate it. On the other hand, the notorious Heartbleed flaw, which lay dormant in the widely deployed open source security software OpenSSL for many years, shows that open source is no panacea. Fortunately, major open source projects made significant, externally verifiable, process changes to prevent something like Heartbleed from happening again. For instance, companies working with the Linux Foundation now collaborate to support the developers working on OpenSSL as full-time contributors.

At any rate, security in these highly networked times has become much more complicated than hardening your system. You can assiduously install all patches and avoid opening suspicious attachments but still download malware from a compromised website you visit or fall victim to a phishing scheme. Security must always be viewed holistically. It is cultural and policy-driven as much as technical.

Contributing to Open Source Projects

If you use open source code, you will get much more from the project by contributing back, both through code and through other practices such as sponsorship and donations. It makes sense to begin by creating an open source project around noncritical software such as a tool in your support software, a platform add-on, or a plug-in.

This section covers making contributions to existing projects, and the next section looks at the extra steps involved in launching one of your own.

Establish the "Why" Throughout the Company

You understand by now that adopting open source is not a matter of copy and paste but represents a serious commitment from your organization. Managers must approve the use of resources for the open source project, including the kinds of participation in the community discussed earlier. You need a story that justifies the investment of checking the code and community and then participating as community members. Thus, you need to explain clearly how the use of the code contributes to the business goals of the organization, and you need to check regularly to make sure that the local goals of the developers stay aligned with these larger goals. A whitepaper from Mozilla and Open Tech Strategies offers ten different overall strategies for creating and running open source projects. The variety of governance structures and motivational frameworks discussed in that paper and the impacts they have on outcomes show how important such choices are.

Hire from the Community

A great way to jump-start your use of an existing open source project is to hire qualified contributors who are currently working on the project. Some companies recruit the leaders or key committers, paying them to work full time on the open source project. Other companies strike some kind of balance, allowing a developer to work part time on the open source code and the rest of the time integrating the code into the company or doing other company-specific projects.

The balance between internal and open source work can be difficult to maintain because managers are always eager to get more help for internal projects and are liable to slip more and more of this work onto the developer's schedule. To recruit developers from the community, your company must show your sincere commitment to the project, through your business plan and your participation in the project. You might also need to review your employee contract and amend it to support developers who want to contribute to their open source projects in their own time. When developers are hired, both management and the developers themselves must stay alert and stick to the original deal about how they divide their time.

Regardless of how much time you invest in the open source project, a healthy project that you adopt for the right reasons will pay you back handsomely for the reasons stated in "Why Are Companies and Governments Turning to Open Source?" on page 2 earlier in the book.

Develop Mentoring and Support

Your employees will need training, both to work with the new code you are handing them and to join the open source project. If you don't already have employees comfortable with working in an open source environment—and it's worth polling your developers to find out, because you might be surprised to learn that some of them have experience with open source—it's worth hiring developers who do, as explained in the previous section. These developers should also mentor others, because this is a key part of bringing developer resources into open source projects, whether within the company or in the wider community. As mentioned earlier, working on open source projects is an excellent way to improve a company's developer skills.

Set Rules for Participation

Open source changes every organization that adopts it, even just to participate in one outside project. Developers at many levels of the organization, and sometimes other employees, are now exposing their conversations and decisions to public scrutiny. Assume that what a developer says in an online forum for an open source project will be seen by the entire world and will last online forever. This might sound like a reason to discourage participation, but it's not. It just means that you need to establish rules for online participation, such as a code of conduct that enforces honest but respectful dialog and that lays out the rights of people who regularly experience discrimination or abuse. Many open source projects provide good codes of conduct that you can adopt. Several codes that have been carefully tested and vetted already are offered by the TODO Group. Revised versions of this code of conduct are implemented by companies such as Amazon Web Services across their open source projects.

There has recently been a heightened awareness of practices in the computer industry (and other parts of society) that discourage the contributions of women and minorities, or that create a harassing and unsupportive environment. These are habits you'll want to root out of your organization, regardless of whether they're exposed to public scrutiny. The company and community must take positive and affirmative steps at diversity and inclusion.

Foster Open Communication

Beyond showing respect, your developers need to engage with the open source community productively, which might require a culture change. Developers might be used to calling an informal meeting of two to four people and making design decisions informally. Some kinds of Agile and Scrum methodologies encourage this behavior, but they also provide techniques for accepting input from wide swaths of the organization. Therefore, Agile and Scrum can be adapted to open source. In fact, open source is open to a wide range of methodologies.

Developers must now learn to conduct all discussions of significant design issues in forums where the decisions are archived for others to view. For changes that you intend to contribute back to the community, discussions must be on the community mailing list or communication channels everyone can join, such as Slack, Gitter, Stack Overflow, or Google Groups. Otherwise, you will take the community by surprise when you submit changes and will probably fail to get changes accepted.

In addition to providing information in a timely manner, developers must also look beyond the narrow horizons of their team and recognize the value of input from a diverse range of people, either from other parts of your company or from outsiders who might live halfway around the world. This is where techniques of respectful dialog must enter your corporate culture. Developers need other new skills, as well: they must be good listeners, take feedback from a variety of developers, reject unsuitable contributions while encouraging the contributor to learn and try again, and deal with people from different backgrounds with varying language skills.

When developers are accustomed to making design decisions in hallway conversations or over the telephone, they might need some time to learn to make decisions in more open ways and to use the less-intuitive communication media provided by mailing lists and bug trackers. Joining an open source project is an excellent way to raise the priority of good communications, honest disagreement, and open, respectful dialog throughout your organization.

Decisions can take longer when the crowd become involved. If you short-circuit the open source discussion process and make a quick decision in order to get a product out the door, you might need to roll back some of the work done later so that the code can be maintained and can adapt to future needs.

An important side benefit of diligently recording decisions is that the project becomes less dependent on particular individuals and can reorganize itself gracefully if key individuals leave. No single person has unique or irreplaceable information.

Launching an Open Source Project

As your organization comes to recognize the benefits of the open source development model, you might decide to start a project of your own. Careful planning can make the difference between a quickly forgotten fiasco and a thriving, sustainable code base. A Linux Foundation guide offers more details about the process and a comprehensive project launch checklist. All the principles of the previous sections apply, and some further considerations are summarized in this section.

Choose a License

Your legal staff must understand how you plan to use the code, including whether you will make closed enhancements for internal use. The license you choose will control your own use as well as the decisions other companies and individuals make to adopt your software. So, make sure it is aligned to the business's strategy.

There is no reason to get fancy, though. Earlier in this book, we mentioned a few of the most popular licenses. Although there is a long list of open source licenses at the Open Source Initiative, we strongly urge that you to go with one of the popular and recent licenses. GitHub offers a guide with a selection of well-crafted licenses. If you don't think one of these meets your needs, go back and candidly reevaluate your motivation for going open source. If you can't use a popular license, you are probably trying to do something out of sync with open source principles, and could end up driving away the people whom you want to attract.

Open the Code Right Out of the Gate

It's best to create a public repository and invite participation from outside your company before you create a single line of code. Otherwise, you won't be able to open your code until you review it thoroughly to strip out proprietary secrets and embarrassing comments. Opening the code from the start—part of an "open source first" approach—will encourage your own developers to follow best coding practices.

Select a name for your project that will resonate with the community. Names can be very personal (for example, Linux was named after the creator of the operating system) or can describe the software, as in the office software called LibreOffice. The project can be named after a mascot or even be a nonsense word chosen to be easy to remember (such as Hadoop). Run through the same checks and due diligence that your legal team does for trademarks to ensure that no duplicate exists, that the name is not offensive in some language, and so on. See the section "Keep Up Communication" on page 22 for information on promoting your project and brand.

The first code you open could be messy, if it's created by people used to internal team work instead of open source development. Opening immature code can be difficult for both your developers and your managers to accept. But simply document the progress you've made and what you need from the community, and you will be rewarded by them—given, that is, a commitment to winning over and mentoring new users. If you have a good idea, many hands will reach in to fix your problems. If you don't have a good idea, you'll hear that unpleasant news from outsiders and will be able to abandon the project before wasting more developer time.

Use Best Practices for Stable Code

You can use one of the popular public repositories to store code and documentation, or can set up a version control system of your own with public access. Have developers check in their changes daily or as often as needed. Developers refer to this practice as "release early, release often." Continuous integration and regression tests (both considered best practices in open source communities) should ensure that the developer doesn't break anything. Most projects still offer formal releases in order to guarantee stability (especially to major corporate users), but open source permits feedback and improvements on a continuous basis.

Set Up Public Discussion Forums

You can't expect people to respect your process and make contributions if you hide decisions from them. As explained earlier in "Foster Open Communication" on page 19, developers within your company are now part of the wider community that hopefully will flock to the project outside your company. A simple mailing list might be all that you need. Any tools you use for discussions should be open to the public and should be based themselves on free and open source software so that you don't put up barriers to participation.

Make Life Easy for Newbies

Attracting people who are unfamiliar with your code and organizational setup is critical at the very start of your project, and it remains important even after you are well established. Do not drift into an insular culture understood only by people who have participated for several years. Devote resources constantly to activities that draw in new people, such as the following:

Documentation
: This ranges from quick *Getting Started* guides to architectural descriptions that explain what is unique and valuable about your project. Many people who lack the skills to contribute code would be happy to write documentation; you need to find, recruit, and engage them.

Conferences and code-a-thons
: These demonstrate your commitment to the community and foster enthusiasm. Many people become long-term contributors after attending such events, which are a good way to recruit new contributors and inspire existing contributors to do even more. They also ensure that the wider community is heard when key decisions are made. Get community involvement in organizing the events as well as attending them. Local meetups can be cheap to organize—offer a space in your offices and buy a few pizzas and sandwiches—and can build strong community support.

Code of conduct
: Establish respect as a key value of your project from the start. A written code of conduct is critical, even if what it says seems obvious to you. Make sure that project leaders intervene quickly when people are rude or abusive. Even a single tense exchange can drive away a substantial number of users. If you are not welcoming to diverse genders and other groups, you will lose (perhaps forever) the chance to recruit from a big group of talent. And the bad reputation will stain your organization, as well.

To-do lists
: When people have been using your code for a while, they begin to ask how they can help. Provide a prominent list of tasks that you—and other members of the community—have identified as priorities.

You need to assess at different stages in development how much time you can dedicate to the community; hopefully others will start to pick up the task and answer questions. Ultimately, though, you'll lose users unless you take their needs seriously, support them in creating the features they want, and give them a say. Hiring a community manager is a good investment: such a member of your staff can educate both company employees and outside community members about how to help the project progress smoothly and productively. As the project grows big and widely adopted, it might become time to hand control over to an independent governance organization altogether, which is the topic of an upcoming section ("Release the Project to an Independent Governance Organization" on page 24).

For an in-depth discussion of how to work well with a community, the book *The Art of Community* by Jono Bacon (O'Reilly, 2012) is very useful.

Keep Up Communication

Talking with a community goes beyond public relations, which typically focus on press releases about major events. You want the community and the world at large to know about evolution in the project before, during, and after each step. Encourage your employees to blog and use social media in appropriate ways to get the news out. Consider a commitment to recruit an informative post at least once every two weeks from someone in the community (and even more often for large projects) along with regular tweets. A small investment in branding, such as stickers that community members can put on their laptops, or socks and t-shirts, shows pride in the project and gets the name where it is seen by the people you want. Such practices attract new users and remind existing community members that you have a vibrant project.

Adopt Metrics and Measurement

We are a data-driven society. All organizations must learn how to collect useful metrics and educate employees on how to use the data when making decisions. Some metrics are easier to collect than others, so you need to determine what's really useful to you. Begin by collecting lots of metrics; then, over time, as you find out which ones are really useful, you can scale back. Typical metrics include the following:

- Numbers of contributors and contributions, and what people and places they come from
- Number of users, which you might be able to calculate roughly from statistics on downloads, mailing list participation, and other proxies
- Growth or shrinkage of the contributor mailing list
- Numbers of reported issues, bugs, and fixes
- Number of forks and stars on GitHub
- Page views of web pages, blogs, and tweets associated with your project

The CHAOSS Community is defining metrics that are useful to collect across most projects.

Generally, you want to see the measures increase. Even an increase in reported bugs can be a good thing because it shows that the code is useful. The speed with which reported bugs are fixed can be a more important metric. If pull requests stagnate or go down, you need to think of ways to promote the project—or perhaps it's time to launch an effort to add new features that make the project more appealing.

If most of your contributions are coming from a couple of organizations, you might need to encourage more diversity. There is a risk that your code will be optimized for one or two major users, losing value for other potential users. And, if a major contributor suddenly pulls out, you can be left without crucial support.

Different metrics are useful in different circumstances. For some projects, it's all right for pull requests and community participation to stabilize. Perhaps your project has a narrow application but is very useful for the people who need it.

Your measurements can become part of a continuous improvement process. Make them available to managers through dashboards and encourage managers to pull them up at meetings and during the process of prioritizing future work. As with nearly all software, good open source tools exist for dashboards and visualizations displaying metrics. Bitergia offers open source dashboards, and Amazon has released an OSS attribution builder and OSS contribution tracker that their OSPO uses to manage its open source projects.

Release the Project to an Independent Governance Organization

Suppose that you have followed the advice of this book and the resources to which we've pointed you. Your project looks like a success and is being adopted by people outside your organization. When the project is big and stable enough, it's probably valuable to make it independent from your company. This will further encourage other organizations to support it, financially and otherwise. It will announce to the world that the project is sustainable and does not depend on your own management decisions for its future, which in turn will draw more people to use it and contribute to it. But because making an independent foundation is a lot of work, you should wait for clear evidence that it's important; for instance, requests from major contributors or the need to raise funds outside your own organization.

Setting up a foundation is a complex task. A few major projects set up independent foundations, such as Linux, Mozilla, and OpenStack, but the vast majority of open source projects—even such popular tools as the Spark data processing tool—work under the auspices of an existing foundation. The Apache Software Foundation, Eclipse Foundation, and Linux Foundation sponsor wide varieties of software that extend beyond their original missions. Other organizations serve particular industries, such as HL7 for health care and Automotive Grade Linux for software in cars.

Open Source and the Cloud

The move to open source during the past decade or so has been paralleled by the adoption of cloud computing at many levels: infrastructure, data, and services. In fact, free and open source software is a major driver of the cloud, and service providers—many of them listed at the beginning of this book—are major creators of open source software. Cloud providers rely heavily on open source software such as Linux, and virtualization software such as KVM and Xen. Customers running their software in the cloud choose open source software for the same reason.

The key advantage open source software offers both cloud providers and cloud users is its cost-free deployment. You can start up 10 instances of a virtual machine, expand quickly to 30 to meet peak needs, and then shrink back to 10 without trying to keep track of cost per seats, or adjust payments.

Open source software has become a *lingua franca*, widely known and deeply understood by experienced developers. This increases its appeal to cloud providers and customers, because they understand the impacts of using each project. Also, basing a cloud business on well-tested, preexisting software allows you to spin up faster and add more enhancements. For instance, most cloud providers remain competitive by adding all the latest hot technologies such as deep learning. The providers realize that making it simpler to operate open source tools has tremendous value to their customers. The most successful new projects are naturally great candidates for new services. The wealth of high-quality open source options in these areas allows rapid upgrades to services and quickly enhances their platforms for customers' development efforts.

Customers also feel safer when cloud providers use open source tools. It reduces the risk of lock-in and allows customers to adopt hybrid solutions that run their applications on multiple cloud providers or using on-premises software as well as cloud providers.

Because cloud providers own and manage their services, they are empowered to release tools and support software as open source and thus benefit from communities that form to improve the software.

Conclusion

The production and use of open source software has matured tremendously over the past decade. From informal collaboration, often around a "benevolent dictator," it has evolved into a discipline. Community managers, open source program offices, codes of conduct, and other facets of organized development practices are widespread. Websites have become more sophisticated, good communication practices and processes are codified on collaboration sites such as GitHub by groups like the TODO Group, and projects recognize the importance of that long-neglected cousin to source code: documentation.

This book described open source as it is conducted by the most advanced, technology led companies and government agencies in 2018, with a look toward the future. We have consolidated resources and references to materials that will make your open source ventures successful and answered questions where you might have had concerns. Yes, there's a lot to learn in the adoption, use, and release of open source code. Yet many companies are doing so successfully. They maximize the strategic value of adopting open source through culture change and by investing in support for developers and all employees engaged in these efforts.

Thousands of companies use open source software, and many contribute to it. This book showed you how to start your own journey with a pilot project, working with developers and other key stakeholders. Poll your developers to find out whether they are already using open source code. It's important for the members of your organization to learn from their experiences and perhaps to involve them in a more formal policy regarding open source. The community will be a willing mentor as you embark on this journey.

Read some of the documents to which this book points and follow some basic good practices for checking the quality of the open source project. Document what you do and use your experience to determine your next steps.

With adjustments and revisions to your corporate practices, you can use open source libraries for such things as deep learning or web development. The big next step is incorporating open source software into your products. An even bigger step is to open your own code and start a new open source project. This book offered an overview of the tasks facing organizations that undertake these efforts, along with pointers to more detailed sources of information.

Large corporations' embrace of open source demonstrates that it is a fixture of software development and becoming the new normal. We can see that software is changing markets and driving the value of all sorts of organizations, ranging from governments to financial services, and including traditional fields such as agriculture and construction. Each organization chooses a balance between building its own software, purchasing closed-source products or services, and consuming or creating open source.

Startups also recognize the power of community. For them, the open source processes are a multiplier for their limited, precious resource of developer time.

Similar principles apply in other areas of creative production (although each type differs in the details): open source hardware such as Arduino, artwork released under some Creative Commons licenses, data provided under an open license, open standards (such as the Open Standards principles defined by the UK government), information provided through an open license by governments, and so on.

Thus, open source software provides value across many fields. It's time to incorporate the best of open source tools and methods into your company strategy and culture, which will increase your competitive advantage.

Furthermore, adopting open source practices—including InnerSource for software you don't want to open to the world—can make your organization more productive, your innovation faster-moving, your employees happier, and your decision-making more efficient. These are the extra gifts of open source you will come to appreciate during your journey.

About the Authors

Andy Oram is an editor at O'Reilly Media, a highly respected book publisher and technology information provider. His work for O'Reilly includes the influential 2001 title *Peer-to-Peer*, the ground-breaking 2005 book *Running Linux*, and the 2007 bestseller *Beautiful Code*.

Zaheda Bhorat is the head of open source strategy at AWS. A computer scientist, Zaheda is a long-time active contributor to open source and open standards communities. Previously, she shaped the first-ever open source program office at Google; launched successful programs, including Google Summer of Code; and represented Google on many industry standards executive boards across multiple technologies. She also served as a senior technology advisor for the Office of the CTO at the UK Government Digital Service, where she co-led the open standards policy, which is in use by the UK government on open document formats.

Zaheda was responsible for OpenOffice.org, and later NetBeans.org, at Sun Microsystems, where she built a thriving global volunteer community and delivered the first user version, OpenOffice 1.0. Zaheda is passionate about technology, education, open source, and the positive impact of collaboration for social good. She serves on the UK Government's Open Standards Board, which determines the standards government should adopt. She also serves on the board of directors of the Mifos Initiative, an open source effort that is positioning financial institutions to become digitally connected providers of financial services to the poor. Zaheda speaks internationally on topics related to open source and social good. You can find her on Twitter *@zahedab*.