O'REILLY®

Compliments of instaclustr

The Benefits of Open Source and the Risks of Open Core

Understanding the Differences Between the Two Models

Danese Cooper & Andy Oram

REPORT

instaclustr
by NetApp

HARNESS THE POWER OF OPEN SOURCE

Build and scale reliable applications faster

Expert consulting and global support

- ✓ Migration to open source
- ✓ Open source strategies
- ✓ Heath checks
- ✓ Technology kickstarter package
- ✓ Solution architecture
- ✓ Operational review

Managed platform for open source technologies

Deploy, manage, and monitor all components of your data layer and related infrastructure

kafka. Cadence® cassandra® APACHE Spark™ APACHE ZooKeeper™

The Benefits of Open Source and the Risks of Open Core

Understanding the Differences Between the Two Models

Danese Cooper and Andy Oram

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

The Benefits of Open Source and the Risks of Open Core
by Danese Cooper and Andy Oram

Copyright © 2022 O'Reilly Media Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Rachel Roumeliotis and Jeff Bleiel
Production Editor: Jonathon Owen
Copyeditor: nSight, Inc.
Proofreader: Justin Billing
Interior Designer: David Futato
Cover Designer: Karen Montgomery

March 2022: First Edition

Revision History for the First Edition
2022-03-10: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098117467* for release details.

978-1-098-11746-7

LSI

Table of Contents

Preface

The open source movement has taken center stage in software development, and its influence echoes through other areas of life such as open culture and open data. Many software companies hope to cement both their revenue sources and their status in open source communities by offering a mixture of open source (also called free) and closed (proprietary) software. The combination is generally called *open core*.

The variety of strategies adopted by software companies, and the use of similar terms for very different things, make it hard for many software users to understand when they have access to open source versus open core; therefore, the clients of open core companies may be taking on risks they don't understand. Indeed, the companies themselves are taking on risks in an open core strategy. This report provides an overview of open source and open core along with a discussion of what makes each practice attractive and where the risks are.

The authors are not neutral in this discussion. Despite the widespread adoption of open core software by many companies, we find that it tends to have negative long-term impacts on vendors and customers alike. On your journey through this report, you can form your own judgments.

CHAPTER 1

What Is Open Source and Why Is It Popular?

Effectively, open source is a model for software development that allows everyone to use software without restrictions, adapt it to their needs, and share their changes with other people. The concept of open source also involves a constellation of practices:

- Communities of programmers and users, often spanning the globe, who collaborate to improve the software
- Transparent communications, fully documented in open archives
- Software licenses that guarantee the rights to use, change, and share the software
- Public, inclusive review processes
- Living repositories that preserve the entire history of a software project's development, including discussions about choices made

Strictly speaking, most of the items in this list are not needed to call a project open source. Although modern open source projects focus on processes—building community, adhering to healthy governance practices, achieving stable funding, etc.—technically, any time a developer releases code under an open license, it's open source. The Open Source Initiative (OSI), a nonprofit set up by leaders throughout the worldwide open source community, has approved many such licenses. Most of those licenses also pass muster with another key organization that was one of the earliest and most influential leaders in this movement, the Free Software Foundation (FSF).

Some open source licenses are short and simple, whereas others are longer and strive to prevent many possible problems that have been seen over the decades. But most don't come close to the complexity of the licenses that normally accompany closed software, which often incorporate surprising and cumbersome restrictions on users.

Still, the previous list of practices applies to most important open source projects today. We will see in the course of this report that the first two items in the list, *communities* and *communications*, are central to the success of open source and tend to be problematic in open core.

How Open Source Changed the Software Industry

Open source historically reflects an intentional return to an earlier ethos in the technology industry. During the initial rise of software, most practitioners were either scientists or hobbyists who freely shared information and innovation as the norm. Academics, engineers in the field, and software hobbyists took for granted the exchange of source code through floppy disks, magnetic tape, and eventually electronic networks. These programmers realized that it took the combined work of all practitioners to debug and co-invent their ideas.

By the time the concept of open source arrived, however, sharing and collaboration in computing had been overshadowed by the tremendous commercial value of software, engendering fierce competition over prices and features. Software and hardware manufacturers had discovered the power of vendor lock-in, a clever and ever-evolving set of practices that make it difficult for customers to switch vendors. Examples of vendor lock-in include physical keys (dongles) or encryption that make data available only to the vendor's program, opaque file formats that may change arbitrarily in new versions of the product and features deliberately designed to differ slightly from the features on competing products.

The vendors still strived for actual customer value, but it suffered in the wake of artificial monopolies created by vendor lock-in. Customers frequently had to wait months to years for bug fixes, even critical security patches, because the vendor would choose to invest its limited programmer staff in other areas such as new features. It was a common practice for companies to make small changes to file formats in order to wring more money from the installed base, by forcing paid upgrades by customers who merely wanted to retain access to previously created documents.

At the same time, as the World Wide Web was driving online commerce, internet servers relied mostly on a proprietary implementation of Unix running on bespoke server hardware; this solution was expensive and a real barrier to new companies getting off the ground. Something had to give.

In 1999, the groundbreaking Netscape web browser had become an open source project called Mozilla, whose best-known product currently is the popular Firefox browser. This release set the stage for open source to disrupt the status quo in the tech industry. Investments from leading companies such as IBM, Hewlett-Packard, and Sun Microsystems helped launch many of the foundational open source project communities, starting with a historic IBM announcement in 2000 that they planned to spend $1 billion on GNU/Linux. Once a powerhouse like IBM was willing to throw their brand behind Linux, other businesses could take Linux seriously.

At the beginning of the open source movement, most established vendors were notably skeptical and removed. They were afraid that the obligations of strict licenses, such as the General Public License (GPL) from the GNU Project, would legally impel them to publish trade secrets from their proprietary code if mixed too closely with open source code. They thought the ultimate success of open source to be unlikely.

For instance, in 1999, the legal department at a major hardware vendor in the server space at that time, Sun Microsystems, published strict guidelines around shipping open source modules. They had to be on a separate DVD from proprietary product executables, with a separate installation to be completed by the end user (not automatically included as part of any one-click installer for setting up a product). Open source components were not to be tightly coupled in Sun's proprietary products (e.g., the software had to work acceptably without any open source add-ons, and such add-ons could cover only nonessential functionality). And above all, Sun would not initiate any project under the GPL family of licenses.

Within less than two years, Sun would release the code for OpenOffice.org under the Lesser GPL (LGPL) because part of the strategy for that project was to entice GNU/Linux desktop distributions to bundle it, and because it needed to work well inside the GNOME desktop interface, which used the GPL license. But although this policy shift happened fairly quickly for a strategic reason, Sun was not so quick to make similar shifts for their flagship software products: the Java programming language and Solaris operating system. It took a full decade from the first proposals to actual open source releases of both products, because even in a progressive culture, change has a very long tail.

Another tipping point in the maturation of open source in the enterprise came when the stranglehold of proprietary Unix, networking, and web technologies broke up under a combination of open source technologies known as the *LAMP stack*. LAMP includes Linux as the operating system, Apache as the web server, MySQL as the database, and Perl or PHP as the programming language for backend servers.

The financial services sector here was the vanguard for a customer revolt. The CTO at Morgan Stanley realized that Linux could be used to drive a farm of relatively cheap commercial x86 servers to provide acceptable compute power that could drive their online business at a fraction of the cost of bespoke hardware and software. This bold move came just as the first dot-com bubble burst in 2000, and companies became desperate overnight to survive by trimming costs. Within a year, all the major fintech companies were switching.

That same LAMP stack fueled the rise of a new type of company, one that offered services on the web. Ecommerce was one such service, but there were many others. Searching the web itself became a service, with Yahoo!, AltaVista, and later Google offering very high quality, zero-cost search engines through a web interface. The "product" that actually drove the search engine companies' success was the capture, analysis, and sale of the data generated by those searches, with market trends and intelligence available for a fee. Experimental business models like this couldn't be brought to life without increasingly cheap hardware and open source software without license fees.

Open source became a democratizing force in the tech industry, enabling new companies to thrive by driving down costs while enabling offerings of better services. Even very proprietary companies such as Apple started using open source. Indeed, an open source project was the basis for the pioneering version 10 of Mac OS (now macOS), driven by the return of Steve Jobs to Apple in the late 1990s. Apple took a version of the historic Berkeley Software Distribution (BSD) and released their core software under the name Darwin. Apple was also choosing to release some of their own products' source code (notably WebKit, an important layout engine used in many web browsers) under open source licenses, because doing that could drive adoption and set up de facto standards.

Sun Microsystems was by now using open source to push their Java agenda. Sun donated the source code for their Java Servlet API to the Apache Software Foundation to create the Apache Tomcat Project, driving mass adoption of Java over Microsoft's competing Active Server Pages technology.

Where Is Open Source Dominant?

If the average computer user—not a computer programmer or dedicated free software user—runs down the list of installed applications, they will probably not see many open source applications. Perhaps the user has Firefox, a leading web browser mentioned in the previous section. The user may also have found it useful to install LibreOffice (descendant of the OpenOffice.org suite mentioned earlier) for authoring documents, slide presentations, and spreadsheets, GIMP for photo editing, or Thunderbird to manage email. Most layout engines—the software that is central to web browsers—are open source, the most important of these being the previously mentioned WebKit as well as Gecko and Blink.

Applications do not tell the real story of open source and free software. If a user could dig down into their operating system, and further into the networking infrastructure underlying the web, they would discover a world of open source. If they decided to try out professional activities such as creating their own software or managing a collection of servers, they would establish an even stronger bond with open source.

In short, open source programmers are particularly good at creating tools for their own use. Proprietary companies charge a lot of money for such tools, and over time it has become clear that the users of these tools—that is, programmers, professional computer or network administrators, AI developers, etc.—do a better job of meeting their own needs. While some companies still offer proprietary solutions in this area, a growing number simply offer open source tools packaged up for convenient use.

Most open source software is hidden from an end user who is creating a spreadsheet or doing ecommerce, but open source is pervasive and makes the end-user activities possible.

Governments have also jumped on the open source bandwagon. In the early 2000s, many governments adopted laws or regulations favoring the use of open source software in government, where feasible. Notable examples include the United States, the United Kingdom, and Peru, although in general the follow-through has not matched the enthusiasm of the initial public flourish.

The triumph of open source can be traced in the evolution of the world's most famous software company, Microsoft. As late as the '00s, Microsoft managers were calling open source a "cancer" and a danger to the software industry. But Microsoft is now one of the top contributors to the Linux kernel and collaborates with open source communities to create tools that work with Microsoft products.

Aren't Web Services Taking Over?

Web services, which you use every time you go online to visit a retailer, bank, or professional service such as SAP or Salesforce, are almost always proprietary. The topic of web services, also sometimes called software as a service (SaaS), lies beyond the scope of this report, but we'll say a bit about it because of its widespread use.

Most web service companies want to fix bugs and add features quickly. They use a variety of processes, such as Agile programming, DevOps, and continuous integration (CI), to allow the quick deployment of new versions of their applications. In other words, when you log in to Facebook (or some other popular service) this morning, you're probably using a different version from when you logged in last night.

This pace is incompatible with the deliberative, collaborative process used by open source communities. A company that adopts such continuous change will quickly lose touch with the open source version of the software and is really using a totally different, proprietary product. Thus, web services are not generally relevant to this report. But a few popular web services—such as the Drupal and WordPress content management systems and the Instaclustr cloud service—provide their services as open source software and work well with communities to update it.

Other reasons for SaaS companies to maintain a firm grip on their software will be explored in Chapter 2.

We're not saying any particular practice is wrong. Continuous updates, handled responsibly, can fix security problems and improve the visitor experience in websites. The CI process can save a lot of time and effort on open source as well as proprietary projects.

Finally, many companies that offer proprietary web services also contribute a lot of developer time and money to open source software, which the companies recognize is superior for their infrastructure. Later chapters will look at this practice.

Why Users Ask for Open Source

The impressive strength of the open source movement, and the popularity of its products, can lead one to ask what makes it so appealing. The benefits of the open source and free software models, listed in this section, help to explain why companies try to ride the powerful wave of open source. People with considerable experience in using and procuring software, having seen the negative impact of proprietary software, tend to look for free and open source alternatives for the reasons cited here.

Avoiding lock-in

This is almost certainly the most compelling reason users adopt open source. Anyone who has used software for a few years has encountered horror stories of lock-in: companies that precipitously raise prices, remove product features that users consider crucial, go out of business without providing an upgrade path to the companies' abandoned customer bases, etc. An open source project depends on contributions, but all users know that the product will be there and will be open to all users so long as enough of them care enough about the product to maintain it.

If a company bases its product on an open standard, the clients might be able to find alternatives (proprietary or open source) without having to change their code or behavior. However, few proprietary products are based entirely on open standards—and if the vendor claims to follow a standard such as SQL, clients will still end up having trouble porting their move code because (as the old joke goes) SQL offers "so many standards to choose from."

Accelerated ramp-up

With proprietary software, businesses have to contact the vendor, negotiate licenses, and manage the software's installation and payments. All these steps are eliminated or streamlined with open source. A few individuals can try out the software as long as they need, without the end of a "free trial" looming over them. They can then spread the software quickly as far throughout the organization as its managers want.

Staff availability

This advantage of open source is often underestimated by its advocates but is acutely noticed by adopters. Learning open source software is cheaper and easier than studying proprietary software (which might force learners to take courses that are expensive and geographically remote). In an industry grievously short of staff, open source makes it easier to find programmers and administrators who know the product, an advantage particularly strong among the most popular open source projects. Job-takers don't have to be trained in an unknown proprietary offering but can work productively from day one with the familiar open source tools.

Freedom from licensing tangles

Proprietary software is usually licensed on some scale. The simplest cost model is "per seat," but modern corporate environments make the calculations more calculated, and SaaS adds even more variables. Such fuss is totally out of place for modern container-based services that scale up and down over a period of minutes. It is also inappropriate for interpreted languages such as Python: a company can't license a compiler on a "per seat" basis because there is no compiler, but no one would want to pay a fee just to run a Python program.

Proprietary licenses often require each employee (or an administrator acting on the employee's behalf) to go through cumbersome registration processes that involve typing in long random numbers. In any case, the counting and monitoring required for proprietary licenses is intrusive.

Adaptability

If you encounter a bug in proprietary software, you have to ask the vendor to fix it, and they will do it on their schedule. They may have other priorities. They may require you to upgrade to a more expensive or complex version of the product to get the bug fix.

With open source, you can hire someone to fix the bug and contribute it back to the core project so all can benefit.

Having a say

Bug-fixing is one obvious advantage of open source, but it only scratches the surface of open source's benefits to involved users. Instead of being at the mercy of a business plan cooked up by a vendor, users in healthy open source projects set the agendas and roadmaps.

If the bulk of the community chooses to go in a different direction from your needs, you can create your own branch of a project and implement the changes you want. Usually, there is a way to accommodate multiple pieces of functionality through modules and options for building and configuring the software. If you need to create your own version (known as forking), you can do that.

Compatibility

Open source developers are used to making their tools work with others. Commercial vendors also want to be compatible with outside tools but are more selective about what they support or how much compatibility they offer. If you use an open source tool and you need it to work with a hardware device, operating system, database or other component that's not yet supported, you can go ahead and add that support.

The reasons cited in this section should demonstrate why so many people insist on using open source for their software needs.

Sustainability and Health in Open Source Projects

To compare open source with open core constructively, it's important to understand the traits and practices that sustain a robust open source project. Leaders of the open source movement have long been concerned that choosing an appropriate license isn't enough to ensure a robust open source project. Some important properties of open source development aren't mentioned in the OSI or FSF definitions, leaving wiggle room for interpretation and intentional or unintentional abuse.

For example, open source projects work best when barriers to entry and open collaboration are removed so that any practitioner can rise through the ranks, collaborate with anyone else, gain reputation, and evolve to participate at any level. Several high-profile, corporate-backed projects have failed because they never managed to achieve this leveling, preferring their own employees' contributions over those of nonemployees.

Overt and implicit exclusionary practices make it hard for women and marginalized communities to participate in open source, just as in other parts of society. During the past decade, most open source projects that develop communities institute codes of conduct to enforce fair and respectful behavior. To realistically attract communities that haven't historically participated much, truly committed projects foster educational projects and other forms of outreach to make the pool of available contributors grow.

Although Apple announced interest in fostering a community around the development of their open source macOS core, Darwin, they failed to follow through with the practices that companies use successfully to create a community.

A similar problem ruined the prospects for VistA, a highly regarded electronic health record software from the US Department of Veterans Affairs. They "threw their software over the fence" from time to time, but arrived too late at attempts to create a community, which would have been critical to render the software useful outside of its original setting in the VA. Several open source teams and companies tried to pull together useful distributions but failed to work together and never managed to achieve critical mass.

In addition to the attributes just described, what are the other considerations that lead to a healthy open source project? Red Hat, a leading provider of GNU/Linux distributions and open cloud solutions, cites these things to consider for a healthy open source project:

Project life cycle
: This measure examines things such as how mature the project is and whether it's attracting new contributors. You can decide from this research how you can best participate and whether the project will last.

Target audience
: A project should state how it expects to be used, and by whom. From this information, you can judge whether the project is meeting the right needs and whether it's right for you.

Governance
: A term with many definitions, governance in this context describes how decisions are being made and how contributors are managed and organized.

Project leaders
: You should be able to identify easily who the current leaders are—and how to contact them. A healthy project has leaders who know its history and uphold its culture and vision.

Release manager and process
: You're seeking here a process for releases that is well thought out, well documented, and managed competently.

Infrastructure
: We've mentioned that the coordination of a community is supported by its tools. Numerous tools support version control, issue requests, builds and releases, communications, and so on. Determine whether the tools on the project are meeting its needs.

Goals and roadmap
: These attributes deal with the long-term direction of the project. The project should define goals and a roadmap and communicate them clearly.

Onboarding processes
: There are many ways to contribute to a project: coding, advocacy, fundraising, documentation, and more. New contributors in all these areas should be supported by documentation and mentors.

Internal communication
: People shouldn't be going off in all directions doing whatever is right in their own eyes. See whether they are communicating, whether decisions are being preserved, and whether communication channels are adequate.

Outreach
: Outreach tries to ensure that users, funders, and others with a potential interest in a project know what it's doing. This task can be hard for many software projects, which can easily become ingrown and focus only on people currently showing interest.

Awareness
: How well do potential users and stakeholders know the project? This attribute is related both to outreach and to target audience.

Ecosystem
: As we've explained, open source projects work with many others. The project should maintain good relations with the projects that it uses or serves and should be a responsible member of this ecosystem.

These guidelines have been compiled from decades of experience at Red Hat and other participants working with open source projects. The guidelines are very helpful to an organization looking to commit to a specific project, whether that organization is new to the open source space or longtime allies of open source. We will later refer to some of these guidelines when looking at open core.

CHAPTER 2
Why Companies Find Open Source Daunting

The previous chapter showed why open source is popular. Vendors want to be accepted by communities clamoring for open source, so they promise an open source solution. But it's harder than it looks to build a company on open source.

The following list describes the main factors that give companies pause. Most of these problems focus on the failure to develop a community around open source. If a company can't develop a healthy developer community, it is left having to fund all development itself and enjoys no gain over proprietary development.

They are working from old revenue models.

The old, shrink-wrapped CD model of software distribution had a simple business model: each user paid for a copy. In some parts of the modern software industry, the pay-per-use model still applies. Enterprise software companies license "seats" and mobile device users pay the Apple App Store or Google Play for a download. But the model is breaking down. For instance, very few teams that develop mobile apps cover their development costs by charging a dollar or two for each download. Often, they offer the apps for free and link them to backend services.

A great deal of ingenuity has gone into deriving revenue from software. You could spend years studying the business models of industry giants such as Google and Facebook. Many companies use the freemium model, giving you a certain set of features and disk space and expecting some users to pay for upgrades. Some other strategies—offering analytics, for instance—could work well with open source software, but others would not. The last chapter of this report looks at models that seem successful over the long term.

They don't want to lose control over the direction of the software.
The previous chapter explained that open source works because users get to determine features and the general direction of the project. One would think that this responsiveness to user needs would be a great boon to a company. Yet many companies start with their own view of what they need to succeed and are loath to cede control to users.

They don't want a competitor to take advantage of their work.
The worst potential example of the previous concern—loss of control—would occur if another company built a better product on the software released as open source. Even a moderately successful competitor could eat into the original vendor's customer base enough to ruin their business.

They don't know how to manage a community.
Perhaps a company is willing to give its community a lot of control over its roadmap. But community management is difficult, a discipline all its own. Open source companies often hire community managers, but the managers might not have enough autonomy or might lack sufficient gravitas in the company to hear the community's needs. A combination of advocacy, organizing, good communications, investment in tools, and other skills go into community management. A failure in this area leaves potential collaborators frustrated, and they don't stay around long if they can tell that the company is insincere or unwilling to fund a healthy community.

Their pace of development makes community participation hard.
As mentioned in the earlier sidebar "Aren't Web Services Taking Over?," many companies want to develop new features quickly. They have internal development processes that put the onus on small, independent teams moving fast.

Open source also thrives on small, independent teams. But the projects usually corral more reviewers and take time for more communication between developers working on different branches. Roadmaps tend to involve more deliberation. The projects move slower than SaaS projects, putting user satisfaction above innovation. A company adopting open source must learn to be satisfied with the pace established by its community.

They need to keep people coming to their site to develop volume.
Companies such as Google and Facebook rely on huge participation, particularly for data collection and analysis. Their strategies have both positive and negative effects that have been extensively analyzed in the press. If your service is based on open source software, people who want to opt out of your data collection (or whatever you're doing) can easily set up a competing site, and you lose a chunk of your potential users.

They lack confidence in their code.
Good software engineering takes a lot of expertise. All across the computer industry, programmers and users suffer from the technical debt of sloppy code. Fast-evolving code bases need careful adherence to standards more than anyone, in order to avoid bugs and maintenance headaches, and yet it is precisely in such fast-evolving environments that developers are most likely to opt for a quick and dirty approach.

Furthermore, many developers follow the risky practice of embedding passwords or other corporate secrets in their source code. Going through all the code to remove these risky details is a big job.

In short, many companies reject open source because they don't want to show the public how poorly they code.

They want to hide something they're doing.
Code might reveal thinking or product planning that the company wants to treat as trade secrets. There also might be less commendable reasons for hiding what they're doing. Privacy violations and other abuses are rampant in modern corporate coding. If companies decide to release some code as open source, they risk that users will find out what's happening behind the scenes, leading to bad public relations and canceled accounts.

Their developers are egotistical about their skills.
Sometimes managers are willing to let communities contribute code, but their own developers reject the contributions. Some

> teams even write their own code to reimplement features that customers submitted. The developers in the company may say that outside code does not meet their coding standards or quality measures. And that may be true but rejecting contributor code is still a bad practice.
>
> First, in a healthy open source community, people who know the community's standards invest substantial time in mentoring other people. Outsiders can progress up the contribution ladder, contributing bigger and bigger changes, and eventually becoming core committers who can approve other people's changes. If this is not happening, the company is open source in name only.

Companies who suffer from these fears might still jump on the open source bandwagon for several reasons: they see how quickly the software is adopted, they recognize that end users and developers trust open source, and they hope to get free donations of bug fixes and even new features from a community. But for the reasons cited in this chapter, some companies are trapped in a proprietary mindset in one or more ways. The most important concerns are to make sure that customers still need the company's paid services, and to prevent competitors from gaining access to those important features. That's why so many try to split the difference through an open core strategy.

CHAPTER 3

What Is Open Core?

Open core consists of proprietary code mixed with open source code, usually provided in a convenient distribution or package for pay. The open source part is often labeled the *community edition*, demonstrating that the company recognizes the value of building a community around the project. (Terminology shifts over time, though.) The community edition is free for download and might be released along with tools that open source communities typically use to manage their projects. In order to meet the vendor's goal of attracting users, the community edition should be a coherent piece of software that performs a set of the desired operations, sometimes rich in features and sometimes lacking.

The proprietary features are released separately with a proprietary license and have often been sold under the label *enterprise edition*. The name suggests that you had better use this edition if you run a business that depends on the software. Vendors can be very sophisticated in evaluating community and enterprise needs: for instance, the enterprise edition may add useful scheduling, modeling, analysis, security, high availability, and other tools that enterprises want.

Because (as the term *open core* suggests), the open source community edition contains a lot of core functionality, it is attractive to academic environments, tinkerers, learners, and small businesses that can get something from its functions without the enterprise features. The vendor hopes that community members will contribute to the open source edition, thus increasing the value of the whole product. Furthermore, the open source edition can furnish trained staff that enterprise customers can hire, and small businesses may switch from the community to the enterprise edition as they grow.

Open core is not a new idea—in fact, it has an extensive history. As early as 1999, in his "Magic Cauldron" paper, Eric Raymond described several business models in which open source could furnish a component. Open core fits into Raymond's "Loss-Leader/Market Positioner" model, wherein open source is used to entice customers to try a product at no cost, along with paid add-ons or other upsell opportunities that are proprietary. Around 2000, Richard Stallman, founder of the Free Software Foundation, suggested a related hybrid model to Sun Microsystems for Java.

By 2004, the buzz around open source forced procedural changes on the Apache Software Foundation (ASF), the leading sponsor of open source projects both then and now. They discovered that their reputation for hosting true open source projects was being compromised by prospective project donors who made splashy announcements about pending contributions but never followed through or failed to change their way of working if they did actually contribute the code.

The ASF thus moved to protect their brand by limiting its use to board-approved "top level projects." They also created the Apache Incubator, which required all donations to enter an incubation phase. During incubation, the original contributors had to clear away legal encumbrances, learn to use "the Apache Way" to build a multiparty community, and successfully release a stable version incorporating changes donated by that community before they could apply to become a "top-level project" and gain the right to use the Apache brand.

Thus, the ASF has continued to champion open source without being diverted from that path by companies trying to capitalize falsely on its brand. Branding and trademarks are routinely used in open source projects now to prevent fraud and the dilution of terms, particularly in famous projects such as Java and Linux.

Most of the new projects coming to the ASF were from large firms with very deep pockets. Smaller startups that wanted to engage in open source were often discouraged from doing so by their funders. Venture capitalists (VCs) are still struggling to modify their 1990s expectation of "10x growth in five years."

Early investors in open source companies often tried to split the difference by requiring some elements of the products they funded to be relicensed and make some of the code proprietary in order to "shelter value." These same VCs encouraged adoption of hybrid business models, one of which was open core. The term *open core* was formalized by Andrew Lampitt during the downturn of 2008, when pressure to "shelter value" was rising.

But curiously, the term *open core* remains ambiguous. Investors with commercial interests in open core software companies often try to redefine the term or broaden its meaning in confusing ways.

A recent video from a roundtable reveals that managers tend to see open core on a kind of spectrum or even matrix (the term *orthogonal* got thrown around) and disagree about the definition. The term has recently been reoriented by Joseph Jacks, founder of the Open Core Summit. Jacks applies open core to "any enterprise that is dependent on open source" and equates it to another term, commercial open source software (COSS). Unfortunately, Jacks's usage threatens to dilute the term to the point where it loses any value. We saw in Chapter 1 that most organizations do professional computer development and administration with open source tools, and one could go down a rabbit hole trying to determine what makes a business "dependent" on them.

Many people have built proprietary software on open core platforms and tools over the decades; this does not compromise the open source projects. It's extremely common to write a proprietary program and compile it with an open source compiler, write a proprietary plug-in for an open source browser, run a proprietary ecommerce site using an open source database to store your data, and so forth—all legitimate practices that preserve the integrity of open source software. Open source, open core, and fully proprietary companies all do these things. The relationship becomes a problem when it bends the open source project to the proprietary needs of the vendor, as described in this chapter. That is the open core we warn about.

Risks of Open Core for Customers and Vendors

Open core, when presented honestly, is just as legitimate a way to derive revenue from software as the strategies of proprietary vendors. Some companies, unfortunately, are not so honest. They may provide a community edition that isn't really viable, or suddenly stop support for the community edition in the hope of forcing users to upgrade.

We think that many open core companies are sincere and want to treat both community and enterprise users well. And because so many companies have been trying open core, the model must be working for some companies for at least a time. In particular, if a project is small, with a niche audience, the community might appreciate and support an open core approach.

Nevertheless, observers in computing have seen the risks of open core for both companies and their clients (see for instance a widely read posting by Simon Phipps, a longtime leader in open source). Open core might not deliver the benefits for which open source is well known.

First, customers who move up to the enterprise edition lose the benefits of open source. They suffer from lock-in and lose their right to determine the project's roadmap.

The biggest risk of open core is for the company to neglect the community edition. The companies adopting open core, as explained earlier, are trying to preserve business models based on a proprietary mindset. Because they are deriving revenue from the licensing of the software, the result is predictable: they will put more and more of their investment in the enterprise edition and eventually abandon the community edition.

Ideally, outside developers would continue to enhance the community edition and give the company leeway to develop the proprietary features—increasing functionality for all. But this happy outcome depends on a lot of unlikely things going well; long-term success is restricted to a small set of the most popular projects. Companies with an open core approach generally begin with honorable intentions but walking the tightrope between community stewardship and commercial pressures is challenging.

The company usually feels that it needs to maintain control over the core in order to ensure that it supports its enterprise edition. Contributors see good features being rejected and leave the project in frustration. As described earlier, companies might cite all kinds of seemingly legitimate reasons for refusing to trust community contributions, but underlying the justifications is fear of losing control.

What if a robust, effective community does develop? In that case, the benefits that led to the dominance of open source come into play. The community can surpass the company's offerings and make the company obsolete.

This danger is not theoretical; it happened to an open core company called Eucalyptus, which developed orchestration software for building infrastructure as a service (IaaS) platforms. One of their clients, NASA, was large and well-endowed enough to create a big set of enhancements that they offered back to the community. Eucalyptus rejected the enhancements because it had a different plan for evolution, so NASA teamed up with Rackspace and created OpenStack. This open source project quickly drew support from large companies in telecom and other major industries that wanted to introduce virtualization into their own data centers.

OpenStack quickly developed a very strong community, consisting mostly of large cloud and telecom companies (including VMware) and remains a central fixture of virtualization, whereas Eucalyptus entered the dustbin of computer history.

To sum up, companies dealing in open source must maintain the trust and cooperation of their community. If they end up in contention with that community, one or the other wins—and the loser goes away. Open core sets up a fundamental source of contention that has a high chance of leading to the demise of its balance between company and community.

Community members can easily underestimate the risks they are adopting. The company is assuring the community that it is committed to open source. But its heart (and pocketbook) are in the proprietary extensions. People may contribute changes to the open source core in good faith, and then watch the company go off in a direction that the community does not approve of. Or the customers may try out the enterprise edition and find themselves victims of the same strategies used by proprietary companies to lock them in and disenfranchise them.

How Open Core Undermines True Open Source

Open core looks like a good plan at the beginning. It looks like you get to be cool by championing open source, while still "sheltering value" to make your funders happy—best of both worlds, right?

Unfortunately, very few organizations can actually muster the agility required to follow the open core model without either undermining their community or sacrificing sheltered value in an effort to preserve leadership.

Let's say an open core project is wildly successful initially. Lots of people are using the open source loss leader, and enough of them are paying for proprietary add-ons to justify continued development. A thriving community of developers and users appears.

What typically happens next is that some members of the community start clamoring for the project to release the code for some or all of the proprietary add-ons under an open source license. If the project fails to do this in a timely fashion, the community may simply reimplement the proprietary features they want in open source, often in ways that cause problems for the project owners. Often the project is forced to comply with community demands vis-à-vis proprietary features to forestall gains in popularity for the community versions. Vendors who give in and open up their proprietary features have to later come up with new innovations to justify continued tariffs. It is a very rare project that can out-innovate a thriving open source community, as we saw earlier with the example of Eucalyptus and OpenStack.

Let's step back a moment and look at the principles that make open source work. It requires a shift in thinking about value creation, because it alters where and how value accretes. For many of the spectacular acquisitions during the past few years, the key factor in the deal was neither proprietary intellectual property nor a revenue stream guaranteed by user lock-in. The attraction that drove acquisition value was nearly always deep and loyal adoption and the influence of the developer/user community around the project. The communities were the focus, for instance, behind IBM's purchase of Red Hat, Microsoft's purchases of LinkedIn and GitHub, and (to at least some extent) Facebook's purchases of Instagram and WhatsApp as well as Microsoft's purchase of Activision Blizzard.

It's a long shot, at best, that a given open core project will be able to navigate community demands, pressure to innovate, and funder impatience while continuing to produce high quality, reliable software. This is the real reason that open core is less valuable than real open source alternatives.

CHAPTER 4

Distinguishing Open Core from Other Trends

We have made some claims fraught with controversy in the previous chapters. We have pointed out that the meaning of the term *open core* is unclear and that both vendors and customers can misinterpret what they are getting. In this chapter, we distinguish open core from corporate or community practices that might seem confusingly similar.

Red Hat

Red Hat is the most financially successful company based on open source, having been acquired by IBM for $34 billion in 2019. Many people deny that Red Hat is truly an open source company. We argue that it has always been and remains one.

Red Hat's flagship product, Red Hat Enterprise Linux (RHEL), remains 100% open source. Conveniences such as easy installation or integration of tools into a dashboard do not impinge on that status, in our opinion. When you run RHEL, you are running a distribution of GNU/Linux, plain and simple.

One of the authors of this report (Andy) does freelance work for Red Hat and can testify to the enormous contributions that developers there are making to "upstream" open source projects. Other companies, of course, make big contributions too, but Red Hat's strategy does not involve extending the projects with proprietary software. They are very up front about basing their cloud services on open source components.

As a case study, Red Hat is unique. That's because they focus on the operating system, a very complex and fussy piece of software with many compiler and hardware interactions and external dependencies. Customers have been willing to pay Red Hat to install and maintain GNU/Linux, whereas one might not find customers willing to pay for similar services in other kinds of software. Red Hat's original Linux strategy has become diluted as it moved "up the stack," making major investments in Java, the cloud, and other areas. But the strategy seems to continue to work for the company, judging from recent reports of a 17% increase in earnings.

MySQL

This is the most popular open source database engine, the critical M in the historic LAMP stack mentioned earlier. Before the company MySQL AB was taken over by Oracle as part of their larger acquisition of Sun Microsystems, it pursued a complex strategy, but one that fit firmly within the open source tradition.

True, MySQL encouraged customers to pay for individual features, and even sometimes allowed a customer to use a feature in a special, proprietary version for a few months. But every feature was soon incorporated into the open source version. The Business Source License (BSL), created by the founders of MySQL, formalizes a delay in the release of software as open source.

MySQL also employed dual licensing. If you ran the database as a separate program next to your web server or other software, you could use the open source license. If you wanted to embed the database into a proprietary offering, you needed to license the database under a separate, proprietary license for which you had paid.

The dual licensing strategy seems particularly apt for databases because many customers would like to embed them in other software. As evidence for this claim, another company known for dual licensing was Sleepycat Software, which licensed a database product called Berkeley DB. (Sleepycat invented dual licensing and got it approved by the Free Software Foundation.)

MySQL has become open core since the Oracle purchase. However, there is still a strong open source offering. The creator of MySQL, Monty Widenius, left Oracle and forked MySQL to create MariaDB, a variant that gained a lot of users. The last we heard, Oracle and MariaDB cooperate on sharing open source code.

Corporate Funding

Most software projects depend on funding from large institutions, whether private or public. The funding does not diminish their open source status. If the license is open source, the software is too.

It's also common practice to take donations from particular customers to fund particular features, and this is totally compatible with open source. As we mentioned in the section about MySQL, we would not deny open source status to a project that does a very limited bit of open core by allowing a company exclusive access to a feature for a short period of time. We think that a project enters a danger zone when it sets up a long-term open core component.

Closed Core

This is an extremely popular practice whereby a company contributes to open source projects that are not central to its business. For instance, major cloud-based companies such as Google create or work on a lot of tools for administration, analysis, and so forth. Kubernetes is a prime example. The term *closed core* was first assigned to this practice in a 2011 blog post by Andy, one of the authors of this report.

Closed core is different from open core because a company is not trying to make money directly from its contributions in closed core and does not layer proprietary features on the open source component. Closed core is one of the actual models covered by the Open Core Summit mentioned in Chapter 3.

Dual Licensing

Dual licensing was mentioned earlier in the section on MySQL. This practice is not a mix of open and closed features, like open core. Dual licensing means that the same software can be licensed (and charged for) in different ways defined by the vendor. The practice is pretty common in software as well as other industries. Depending on the needs of the customer, a vendor can offer the same products under different pricing schemes.

InnerSource

InnerSource is not a licensing model, but an organizational practice inspired by open source. Large companies are using InnerSource to develop internal or proprietary software. The managers and developers study the communications and other methods used by open source communities, described in Chapter 1, and mimic them on internal projects. Sometimes companies employ people who are familiar with open source practices to work on InnerSource projects, in order to benefit from their expertise. But the results are not open source (unless the company decides to open the software later).

Source Available

Many proprietary companies offer their source code to customers willing to pay for the privilege. Unix is a famous example; many people treated it as free software (before that term was invented) and wrote books about how to change and recompile Unix source code. A book published in 1977, *A Commentary on the Sixth Edition UNIX Operating System* by John Lions, drew extensively on the source code, became a runaway success, and remains a computing classic. But Unix was definitively not free software, as academics altering the software discovered when its owner, AT&T, sued them in the early 1990s and when the Linux developers were similarly sued by AT&T's successors.

Making the source available is a convenience to customers who want to check for bugs and perhaps even make local enhancements to their version of the product. But the customers are not allowed to share the original code or their changes outside the organization. The original vendor maintains full control.

Open Hardware Cores

We mention this hardware project simply because the name is similar to the software practice in this paper. OpenCores has nothing to do with this report's topic. It is an open hardware project that collaboratively develops hardware chips and boards and places the designs under open licenses. The open hardware movement complements open source software and is inspired by many of the same principles but is a distinct issue.

CHAPTER 5

Why Open Source Is Better Than Open Core

The advantages that open core vendors promise to their customers, and even the advantages that vendors naively expect to gain from open core, rarely pan out in practice. This chapter lays out some of the reasons that open source is taking over software infrastructure, while open core companies routinely disappoint their customers and don't thrive.

Open Source Mitigates Vendor Lock-in

The proprietary features in open core offerings are limited to the company providing those features, just like any proprietary software. Just as you can't easily move your spreadsheets from one program to another without breaking something, you can't easily leave an open core company for a competitor.

Nor can you port a feature you like from a competing product into the proprietary open core product. In contrast, open source projects share ideas all the time. Many developers who work for one open source project take their code to other projects, even projects commonly seen as competing.

Open Source Nurtures a Healthy Community That Drives Better Innovation

As described earlier in this report, a robust open source project enjoys a diverse community of people who work on it individually or together. The sun never sets on some open source communities. This community may or may not work faster than a team hired by an open core company. If the community can innovate faster, the business model of the open core companies breaks quickly as the community edition surpasses the proprietary edition in features or quality.

But even if a dedicated team of professional programmers can move faster than the community, what the community produces often proves superior in the long run.

First, the community consists of the most intensive users of the software, so their ideas for both features and user interfaces are probably more in line with what the customers want. We're not looking at the movement through rose-colored glasses: we know that open source communities can be exclusionary and can develop destructive factions. Even so, the open source community is less likely to suffer from groupthink than a tight-knit team run by a single company.

Similarly, customers who rely on open source software are likely to venture into more adventurous changes than the development team at the open core company. A company tends to have short-term goals. It may launch a major overhaul or new direction, but probably only one at a time. On a large open source project, several teams can be branching off into wildly different areas, and the successful branches are incorporated into the project.

Open Source Increases Code Quality Through External Code Inspections

Once again, groupthink weakens a product. Companies brave enough to submit their code to outside review can fix more bugs and security problems. Although not all customers have the skills to evaluate software, many will. They are motivated to spend time scrutinizing the code on which their organizations rely. Because the customers' professional staff can fix open source bugs as well as find them, open source projects are less likely to have known bugs that fester unfixed for years.

Security is not ideal in open source, to be sure. The notorious Heartbleed bug in OpenSSL and the more recent security flaw in Apache Log4j proved that. But two points are important to note: First, OpenSSL is a complex, subtle piece of security software requiring specialized expertise to understand. Second, the bug was fixed quickly once it was discovered by researchers. (Replacing existing deployments took longer because it depended on the individual actions of downstream users.)

CHAPTER 6

Is Your Vendor Doing True Open Source?

This report has demonstrated the benefits of open source for companies and their clients alike. But as we've seen, companies try to present themselves as open source while behaving very differently. They may be honestly confused about where their strategies depart from open source or may be deliberately hiding strategies for locking in clients. This chapter helps you separate true open source organizations from those who just masquerade as open source.

Is 100% of the Code Licensed Under an Unmodified, OSI-Approved License?

The license fundamentally divides open source from open core. A company can certainly offer different products or services under different licenses. But the product or service you want to use should be 100% open source. An open core strategy is a red flag that should warn you off. Even if the features that are proprietary are not features you want to use, the company's adherence to the flawed open core model will weaken their commitment to the open source part of the product.

Also, make sure that the company hasn't modified a license they adopted from the OSI or FSF. Any change to the license, even an apparently benign change, signals danger. Even in the best case, the company adding or subtracting a clause is departing from best practices, because the OSI-approved licenses have been extensively examined by the best programming and legal minds. In the worst case, the change to a license reflects a hidden plan to lock in customers or break the company's promises to the community.

Is There a Diverse and Healthy Community Around the Code?

Community participation reflects both a company's commitment to open source and the quality of the product or service.

Look for signs of outsider participation: discussions of the code on forums, bug reports containing source code, and check-ins of code by nonemployees. You should see strong participation if the project is really open source. You should also see contributors who moderate conversations so that arguments are resolved peacefully and people are listened to respectfully.

Is the company accepting code from outsiders? Some companies reject major changes. A company that requires all the code to be written by its own team probably also has a hidden agenda. Regardless of the reason they cite for rejecting outside contributions, the effect is to build up a code base owned entirely by the company. (Some companies also require contributors to assign copyright and ownership to the company, which may represent a legitimate precaution or something more sinister.) At some point, the company could well take the code private. The code that was released earlier can still be used by the open source community, but further changes by the company are proprietary and will form the basis of a proprietary product.

Is the Open Source Project Managed by an Independent Body?

An open source project is more sustainable if it is managed by a vendor-neutral consortium. There are several large foundations who know how to manage projects and can help open source projects maintain their funding, infrastructure, and communities. The Linux Foundation, Eclipse Foundation, and Apache Software Foundation are probably the most trusted. A project working with one of these foundations is on a relatively firm footing, although ultimately its success depends on the skills and dedication of its own leaders.

It's hard to set up a foundation dedicated to one open source project but doing so is preferable to having a company run the project internally. The company in charge of a project is always tempted to distort goals to favor its business needs, although many companies do manage open source projects fairly and successfully.

How Does the Vendor Make Money?

Some ways of funding a company are more compatible with open source than other ways. We'll explore these issues more in Chapter 7.

As mentioned earlier, some companies are frankly proprietary, presenting a service as "closed core," but contributing to software infrastructure projects that are open source. If strong communities build up around these projects, support from proprietary companies is a fine model.

Pure open source is a difficult model, but possible. The largest example is Red Hat, which creates and contributes to numerous open source projects. Red Hat makes money by offering services that range from support for computing environments to providing ready-made cloud solutions.

The cloud solution is probably the most common way to make money from open source directly.

But beware of a company that makes money from selling data from a cloud service or mobile app. The company cannot sustain this business model if other organizations use the software. Selling data is sometimes also abusive of people's privacy, although not necessarily so—some companies do a good job anonymizing data and selling it for legitimate purposes.

Is There a Published Roadmap for the Project?

Because an open source project welcomes contributions from the public, its goals should be clear. It should be flexible enough to handle innovators with other ideas, but its leaders should present both a coherent vision for the future and a specific list of goals. Companies that do things without discussing them thoroughly with outsiders are pursuing a proprietary strategy even if they release the code under an open source license. Healthy community participation is squelched by secrecy.

CHAPTER 7

How to Sustain a Company While Supporting Open Source

We hope that this report has clarified the difference between open source and open core and has persuaded you to adopt true open source as a user, developer, or vendor. What remains is to look at the current state of companies devoted to open source. How are they achieving sustainability?

We don't deny that open source projects have to work hard to get funding. Small ones often creep along on the backs of a few dedicated volunteers who might give up any day. Some observers blamed the Heartbleed security flaw, mentioned earlier, as a problem of insufficient support, and the same diagnosis has been aimed at the recent security flaw found in Log4j.

Still, projects find a way to keep going. We review several current models in this chapter.

Closed Core

This option, introduced in Chapter 4, is the most common source of funding for open source projects. Companies often don't try to run the open source projects; instead, they are handed over to a foundation. But companies can be intimately involved in these projects. In addition to donating money, they can put their own programmers on the projects, deliberate about the future of the projects, and create branches for their own use that they contribute back to the upstream projects. Such contributions help to ensure the continued existence of projects that the companies depend on internally.

Although most companies engaging in closed core are in the software business themselves, they can be in any business. The automotive industry, which is quickly becoming a software industry, supports open source projects.

Cloud Offerings

This business model bases revenue on open source by offering access to one or more open source projects as a SaaS or platform as a service (PaaS) offering. In other words, users are able to download the software and install it on their systems. Many choose to pay the vendor because it's cheaper or more convenient to use the vendor's cloud service. WordPress and Drupal are popular web hosting services offered under this model.

It would muddy the concept of open core to apply it to a cloud offering based on open source software. There is even a restrictive GNU license (the Affero GPL) that blesses this arrangement as free software.

Cloud Services Veer Toward Open Core

A huge number of cloud services offer access to popular open source tools. These are tools that programmers and administrators know and love; tools used for hosting, monitoring, analytics, graphical modeling, and numerous other things. But to be honest, it's hard to find cloud services that are restricted 100% to open source. It can also be hard to assess, just from viewing the services advertised on their websites, whether they also have proprietary offerings. So today, open core is more prevalent than open source in cloud services.

Service and Custom Development

Programmers and system administrators who are familiar with open source offerings—and perhaps developed them—can offer to install, manage, monitor, and customize the offerings. This is a classic way for programmers to make a living, even with proprietary software. Several scenarios fall under this business model:

- Teams that create and maintain the particular open source software they're supporting
- Experts in one or more open source tools
- Consultants with general backgrounds who know open source tools along with proprietary ones

In any case, this solution is mostly for small organizations because it doesn't scale well. A programmer can work on only a certain limited number of customization projects at a time; it requires a lot of experience to be efficient and offer value for services, so the model is not for everyone.

Nonprofit Status

Our final model is to apply for grants as a nonprofit. The MITRE Corporation has been doing strong work for decades as a nonprofit, living off government contracts.

Conclusion

Many companies capitalize on the public relations halo surrounding the popular open source movement without understanding or authentically engaging in open source. They've tried in several ways to redefine open source to defend their market position or lack of a suitable business model, while claiming to be friendly to collaborators. Some have built open source communities around their products without thinking through the implications.

Open source has proven its lasting value. Open core, on the other hand, is a risky balancing act that reflects obsolete approaches to markets and software value. We believe that this report has provided convincing arguments in favor of choosing the real thing.

About the Authors

Danese Cooper is a well-known leader and advocate for open source, with 30-plus years of experience in technology and 22 years contributing to the open source movement. She served for four and a half years as head of open source software at PayPal, Inc., during which time she was the first chairperson of the Node.js Foundation as well as the founder of InnerSourceCommons.org and author of *Adopting InnerSource* for O'Reilly. Previously, Ms. Cooper served as the CTO of Wikimedia Foundation, Inc., as the first chief open source evangelist (and founder of the world's first open source program office) for Sun, and as senior director of open source strategies for Intel. She concentrates on creating healthy open source communities and has served on the boards of the Drupal Association, the Open Source Initiative, and the Open Hardware Association and has advised Mozilla, the Linux Foundation, and the Apache Software Foundation. She also runs a successful open source consultancy which counts the Bill & Melinda Gates Foundation, the SETI Foundation, Harris Corporation, and Numenta as clients. She has been known to knit in meetings.

Andy Oram is a writer and editor in the computer field. His editorial projects at O'Reilly Media have ranged from a legal guide covering intellectual property to a graphic novel about teenage hackers. Andy also writes often on health IT, on policy issues related to the internet, and on trends affecting technical innovation and its effects on society. Print publications where his work has appeared include the *Economist*, *Communications of the ACM*, *Copyright World*, the *Journal of Information Technology & Politics*, *Vanguardia Dossier*, and *Internet Law and Business*. Conferences where he has presented talks include O'Reilly's Open Source Convention, FISL (Brazil), FOSDEM (Brussels), DebConf, and LibrePlanet. Andy participates in the Association for Computing Machinery's policy organization, named USTPC, and is on the editorial board of the Linux Professional Institute.