

The
Pragmatic
Programmers

Business Success with Open Source

Strengthen Your Business with
Free and Open Source Software

VM (Vicky) Brasseur
edited by Adaobi Obi Tulton

Early Praise for *Business Success with Open Source*

We all use open source software, even if we don't realize we do. If you manage or develop products, you need this book. Even more importantly, if you sell products, you need this book. Don't fear open source. Learn how to use it and use it well with *Business Success with Open Source*. Your customers will thank you.

► **Johanna Rothman**

Author of the Modern Management Made Easy series

Almost every business interacts with open source software, but they don't always do it with intention. For those who want to get the most value from open source software, *Business Success with Open Source* is a great resource. It covers all aspects of consuming and producing open source software, all with VM Brasseur's expert and engaging style.

► **Ben Cotton**

Author of *Program Management for Open Source Projects*

Business Success with Open Source is a brilliant tool to help C-level executives understand how to make the most of open source. The concepts are solid, the ideas are exposed clearly; it's obvious that Vicky knows what she is talking about, and that her ideas are battle tested.

► **Thibault Martin**

Director of Program Development, The Matrix.org Foundation

This book should be on every technical leader's bookshelf. VM explains complex topics in a way that is approachable and engaging, and relevant to just about any business.

► **Jill Rouleau**

Principal Software Engineer, Ansible Project

Business Success with Open Source

Strengthen Your Business with Free and Open Source Software

VM (Vicky) Brasseur

The Pragmatic Bookshelf

Dallas, Texas



See our complete catalog of hands-on, practical,
and Pragmatic content for software developers:

<https://pragprog.com>

Sales, volume licensing, and support:

support@pragprog.com

Derivative works, AI training and testing,
international translations, and other rights:

rights@pragprog.com

The team that produced this book includes:

Publisher: Dave Thomas

COO: Janet Furlow

Executive Editor: Susannah Davidson

Development Editor: Adaobi Obi Tulton

Copy Editor: L. Sakhi MacMillan

Indexing: Potomac Indexing, LLC

Layout: Gilson Graphics

Copyright © 2025 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced by any means, nor may any derivative works be made from this publication, nor may this content be used to train or test an artificial intelligence system, without the prior consent of the publisher.

When we are aware that a term used in this book is claimed as a trademark, the designation is printed with an initial capital letter or in all capitals.

The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg, and the linking g device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions or for damages that may result from the use of information (including program listings) contained herein.

ISBN-13: 979-8-88865-049-3

Encoded using recycled binary digits.

Book version: P1.0—March 2025

Contents

Heartfelt Gratitude	xiii

Preface	xvii

Part I — Fundamentals of FOSS

1. Lay the Foundation	3
Why FOSS Matters	4
What Is Free Software?	5
So Then What Is Open Source Software?	8
The Difference Between Open Source and Free Software	11
Community and Contributions	13
The Importance of Standards	14
2. Make the Intangible Profitable with Intellectual Property . . .	17
IP Is the Foundation of FOSS	17
Introducing Intellectual Property	19
Trademarks	19
Patents	21
Copyright	24
Contributor License Agreements	34
Developer Certificate of Origin	35
3. Licenses: The Rules of IP Engagement	39
Basics of Licensing	39
The Contractual Controversy	41

	Licensing Innovation: FOSS's Killer App	43
	Types of FOSS Licenses	44
	What Makes a FOSS License "Open Source"	50
	Summary of Some Common FOSS Licenses	53
	Other Types of Licenses	54
	Derivatives	58
	Distribution	60
	FOSS License Compatibility	61
4.	Avoid Common FOSS Business Risks	65
	Inbound FOSS Risks	65
	Outbound FOSS Risks	75
5.	Strengthen Your Business Through FOSS	87
	Innovation and Digital Transformation	87
	Lower Total Cost of Ownership	89
	Reputation and Brand	90
	Recruiting and Onboarding	92
	Staff Retention	93
	Diversity	94
	Improved Security	96
	Strategic Business Advantage	98
6.	Introducing the Open Source Program Office	101
	What Is an OSPO?	101
	Does Your Company Need an OSPO?	102
	Functions of an OSPO	103
	What Kind of OSPO Do You Need?	107
	Roles Inside an OSPO	113
	Placement of an OSPO Within an Org Chart	117

Part II — Inbound (Using) FOSS

7.	Recognize and Acknowledge Your Inbound FOSS	123
	Inbound Defined	123
	Where to Find Inbound FOSS in Your Company	125

	Inbound Stakeholders	127
	Software Supply Chain	129
8.	Inbound FOSS in Your Corporate Strategy	131
	Role of Inbound FOSS in Digital Transformation	131
	Avoid NIH Syndrome	140
	More Rapid Innovation and Time to Market	141
	Reduce Cost of Maintenance	142
	More Secure Solutions	143
9.	Establish Effective Inbound Policies and Procedures	145
	Why We Need Policies and Procedures	145
	Policies and Procedures Are Business Assets	146
	Make Policies Useful (and Used)	147
	Gather Knowledge and Collaborate for the Best P&P	153
	Deploy Policies Effectively	155
	Enforce the P&P	156
	Evolve the Policies	160
	Clean Up the Clutter	161
10.	Get Started with Your Inbound Policies	163
	Inbound FOSS IP Policies	163
	Inbound FOSS Security Policies	166
	Inbound FOSS Sustainability Policies	168
	Inbound FOSS Integration Policies: Vendors	169
	Other Policies You May Need to Consider	171
11.	Know the Links in Your Software Supply Chain	173
	Supply Chain Defined	173
	The Software Supply Chain	175
	Types of FOSS Acquisition	176
	Software Supply Chain Awareness Is Growing	178
	How to Learn About Your Software Supply Chain	179
	The Process (Roughly)	182
	Tools	192
	Software Bill of Materials (SBOM)	194

12. Maintain FOSS License Compliance	201
What Is License Compliance?	201
What Qualifies as Infringement?	202
License Compliance Enforcement	202
Consequences for License Infringement	204
Basic License Compliance	206
Inbound License Policies	209
Don't Legislate. Automate.	211
 13. Sustain. Maintain. Retain.	 213
Sustainability Is a Business Matter	213
FOSS Sustainability	217
What Your Company Can Do About It	219
 Part III — Outbound (Contributing to and Releasing) FOSS	
 14. Level Up Your Business with Outbound FOSS	 229
What Is Outbound FOSS?	229
Outbound Impacts the Entire Business Organization	229
Outbound Isn't Limited to Code	231
Why Do Outbound FOSS?	232
 15. Outbound Policies and Procedures	 233
Outbound FOSS Policies	233
Outbound FOSS Procedures	240
Postponing P&P Is a Mistake	241
 16. Strategic Benefits and Risks of Outbound FOSS	 243
Be Clear About Your Goals	243
How Outbound Contributions Can Benefit Your Strategy	246
How Releasing FOSS Can Benefit Your Strategy	249
Risks of Outbound	255
 17. Master Contribution Fundamentals	 263
Contributions Sustain FOSS and Your Business Both	263
Ways to Contribute	264

Downstream vs. Upstream	269
Intellectual Property Considerations	273
Verify Your Corporate FOSS Contribution Policies	277
Track Company Contributions	278
Corporate Contribution Metrics	281
18. Develop Your Contribution Strategies	283
Identify Your Internal Constraints	283
Identify Critical Projects	285
Identify Project Needs	285
Define Your Contribution Strategy	286
Ensure Policies and Procedures Are in Place	287
Onboard Outbound Contributors	287
Get the Word Out	288
Start Contributing!	289
The Bees4Less OSPO Contribution Strategy	289
All Big Ideas Start Small	294
Rely on Your OSPO	294
19. The Strategy of Releasing FOSS	297
What It Means to Release a Project	297
Why Organizations Release Projects	298
Release Strategically	301
Choose What to Release	305
Styles of Project Releases	308
Bees4Less and the Open Hive	313
20. Releasing FOSS Projects	317
Collect the Release Team	318
Choose a Release Style	320
Name the Project	322
Prepare the Project for Release	329
Decide Whether to Use a CLA/DCO	336
Track Metrics and Measure Value	340
Choose Release Location	340
Select a License	344
Apply the License	348

Comply with Component Licenses	353
Prerelease Checks	353
Release!	354
21. Engage with Your FOSS Projects	355
Get the Word Out	355
Manage the Community	360
Manage the Project	361
Leave the Project	364
22. Incorporate FOSS into Business Models	369
What Is a Business Model, Anyway?	369
Open Source Is Not a Business Model	372
Making FOSS a Part of Your Business Model	374
It's More About Business Than FOSS	379
Open Core	381
A1. Acronyms & Initialisms	385
A2. Glossary	389
A3. Links from Chapters	405
Preface	405
Part I: Fundamentals of FOSS	405
Part II: Inbound (Using) FOSS	407
Part III: Outbound (Contributing to and Releasing) FOSS	410
A4. Worksheets and Resources	419
FOSS License Compatibility Matrix	420
Outbound FOSS Contribution Planning	421
Choose a Release Style	422
A5. Other Useful Resources	423
Forge Your Future with Open Source	423
Program Management for Open Source Projects	423
Producing Open Source Software	424
The Good Governance Handbook	424
Intellectual Property and Open Source	424
Open Source Law, Policy and Practice (Second Edition)	424
The Business Value of Developer Relations	425
Business Model Generation	425

Index	427

Heartfelt Gratitude

The best part of writing a book is when you finish that last page of content.

The second-best part is getting to thank and praise everyone and everything who helped you get that far.

People Who Helped Me Along

My name isn't the only one on the cover of this book; Adaobi's is as well, and it more than deserves to be there. Her experience, insights, patience, wit, humor, and charm helped get me through the self-inflicted ordeal of writing a second book. It's not exaggeration to say that this book would not exist without her help. Everyone should have an Adaobi in their lives, and I'm grateful that I already do.

Hers isn't the only editorial assistance I received along the way. Both Brian MacDonald and Margaret Eldridge also played critical roles in getting this book across the finish line. Y'all are fantastic, and I'm so glad you're here for me.

Unlike some tech publishers, Pragmatic Bookshelf has a strong culture of improving books through feedback and iteration. The editorial staff is one part of this, but the technical reviewers are another. These folks waded through the entire text and provide feedback, and their help is priceless. Nearly a dozen people were generous enough to donate their time to help me create a better book for you. They are (in alphabetical order by first name):

- Ben Cotton
- Christian Paterson
- Jill Rouleau
- Johanna Rothman
- Karl Stolley
- Kevin P. Fleming
- Lisa Caywood
- Monica Ayhens-Madon
- Stephen Walli
- Thibault Martin
- Uberto Barbini

I owe each and every one of you a hearty meal, at the very least. Thank you.

Innumerable other folks helped me along the way. My dear hearts on Hash-Hash. The Bad at Computers Discord. The Baking Discord. The Knitty Discord. My Twitter friends, until a billionaire turned the site into a cesspit of hate. My Mastodon friends afterward. Everyone at a conference who asked when the book would be out and gave me confidence that maybe this actually is a good idea. My parents for gently asking how the book is going without pressuring me. Percy and Nigel. Osgood and Basil.

If you're not listed here but should be, I apologize. I have a mind like a steel sieve.

I wrote a lot of my first book from airplanes and airport lounges. This one was written almost entirely at home, thanks to a pandemic.

Authorial Coping Mechanisms

I started writing this book using VS Code on Ubuntu Linux. Then I realized that fighting with my tools was distracting me from actually writing words, and the rest of the book was written using VS Code on an M1 Mac Mini. Overkill for a pile of Markdown files? Maybe, but at least I knew my music and Bluetooth would work.

Speaking of music, my *Work - Chill Only* music playlist ought to be listed as coauthor. It was there for every page of this book, though it evolved over time (as all good things should). It picked up some Toumani Diabate, a little *Pentiment* soundtrack, some Andrey Vinogradov, and several other good albums along the way. I tested whether a person can become tired of the works of Jessica Curry¹ and have proven that, no, one cannot. Thank you, Jessica. You've now gotten me through two books.

In all but the hottest months of the year, I wrote while swaddled in a wool shawl that I knit using the Sycamore Creek pattern by Naomi Parkhurst. I love this darn shawl so much. Do check out Naomi's work and support her on Patreon² if you have the means.

Even the worst heat wave couldn't reduce my tea consumption. Gallons and gallons of it went into writing this book, and most of them were from the experts at Upton Tea.³ Thanks, Tea Team. I couldn't have done this without you.

1. <https://www.jessicacurry.co.uk>

2. <https://www.patreon.com/gannet>

3. <https://www.uptontea.com>

You, Yes You

No one writes a tech book because they want to become rich and famous. They write it because they want to lift up others by sharing their knowledge.

You are, therefore, the most important person. Without you this book wouldn't exist. Thank you. I'm grateful for the time we spend walking together through these pages.

Preface

Free and open source software (FOSS) is everywhere and is the driving force behind nearly all software developed today. According to some estimates, it's not uncommon for FOSS components to make up over 90 percent of a modern piece of software. A programmer will take one component—say, one that does authorization—then write some code to integrate it with another that provides communication over the internet, then to one that logs all errors and traffic, and then to yet another that displays a user interface based upon a selection of predesigned templates: dozens or hundreds of FOSS components, integrated and customized to support your company's unique business logic, brought together in days to create software that previously would have taken your company months or years to develop.

How much would you pay for that sort of boost to your company's innovation? Don't answer yet!

What if I told you that along with blazing fast innovation you also get a robust and resilient community of software developers who continually grow, maintain, and support this ecosystem of components? And what if I also told you that right this moment, somewhere in your company, someone is a member of that community and is using FOSS to provide a great deal of business value to your organization. You may not know what they're using or how. You may not know who is doing it. You may not even know it's happening, but I assure you that it is.

Now how much would you pay? \$25,000 a year? \$100,000? \$1,000,000? What would you say if I told you that your company is already benefitting from free and open source software and it's doing it for the low, low price of *free*. That's right, you get the rapid innovation, the caring community, and heck, we'll even throw in increased software security, all for software that's probably not costing you a penny in licensing fees.

That, my friends, is already pretty amazing. But you can do even better.

It doesn't matter what industry your company is in or even whether it sells any sort of software; learning more about FOSS and how to use, contribute to, and release it can be the strategic edge that your company needs. With the proper knowledge and approach, FOSS can form the cornerstone of a digital transformation effort, increase developer retention, decrease recruiting cycles, ensure cutting-edge security, and reinforce the company brand—all this and more, by shifting your company's FOSS strategy from something that's accidental to intentional.

That's why you're here, reading this book. Welcome! You're going to learn a lot of useful stuff here, and when you use it you'll not only strengthen your company, you'll also strengthen the FOSS ecosystem that helps to support it. By the time you're done reading this book you'll have the knowledge to do all that and more.

Who Should Read This Book?

We'll prevaricate and say otherwise in public, but if you corner most authors, you can get them to admit that they'd like *everyone* to read their book. They'd like you to give it as presents at baby showers and graduations, as rewards for accomplishments big and small, and as thinking-of-you gifts to long-lost friends and lovers. That it may be a textbook about orbital dynamics doesn't matter; the more eyes on the work, the happier the author. However, when we return from dreamland into the real world, we sadly confess that, no, not everyone will get value out of our work. This book, alas, is no different.

Since that value is the most important thing here, the question of *who* should read this book is less relevant than *why* someone should read this book. What problems are you, the reader, facing and can this book help you with them? If your answer really is “orbital dynamics,” then this will not be the book for you (maybe you should attend some baby showers or graduations). If your answer is that you need to work with Free Software or open source in some way in your job, then you're in the right place.

If you're faced with participating in a digital transformation effort, or reducing business risk, or building a product roadmap, or creating a company or team strategy, or if you need to round out your existing FOSS knowledge by filling in some gaps, then you'll get a great deal of value out of the book in your hands right now.

While the usefulness of the book is not limited only to this venue (I hope), the expectation is that you're reading it in some sort of a business context. You are a leader of some sort in your company or organization, be it C-suite or

line manager, marketing or programming, entrepreneur or chairperson of the board, strategist or tactician. Your company's business model may not turn on creating and selling software products, but it does rely on software in some way to deliver value to its customers.

In the case of companies undergoing digital transformation, your company may now be experiencing whole new paradigms of software, such as micro-services, containers, continuous integration and delivery, DevOps, and cloud computing. All of these situations depend upon FOSS. Your company and your career will thank you for reading this book and learning about how to make the most of free and open source software in a business context.

Note that one problem I didn't mention is when you, your team, or your employees would like to contribute to FOSS projects. This book certainly will help there, as it will walk you through creating policies and procedures for those corporate-sponsored contributions as well as selecting any projects on which your company would like to focus its efforts. The nuts and bolts of those contributions, however, are covered in my other book: *Forge Your Future with Open Source*.¹ I encourage you to get it for yourself and your team members so they can be more successful with their contribution efforts.

How the Book Is Organized

The matter of FOSS in the business world is larger and more complex than most leaders know. If they're aware of the concept at all, they see only the small part of it that concerns them most: license compliance, security audits, product releases, and so forth. Nearly every company with which I've worked has missed the bigger picture of how FOSS is an asset both to their business and to the world at large. By seeing only the trees and not the larger forest, these companies are overlooking not only huge strategic opportunities but also possibly some potentially company-ending risks.

This book provides an overview of that larger forest as well as some guideposts to help you get through it safely, though of course only you and your company can take the actual journey itself. You'll learn more than enough to start developing a comprehensive open source strategy, focusing on the elements that are most important for the success of your company and lines of business. Those elements tend to fall into one of two buckets: inbound and outbound. Along with the fundamentals, they form the three pillars of FOSS and business. They're also the parts into which the book is organized.

1. <https://fossforge.com>

Part I: Fundamentals

In [Part I, Fundamentals of FOSS, on page 1](#), you'll learn about the definitions of open source and Free Software, the difference between them, and why it matters for your business. You'll also learn the essentials of copyright and licensing in general, the types of FOSS licenses, and what makes them special to you and to all companies. This part of the book discusses some of the potential risks your company may face by using—or, more accurately, incorrectly using—FOSS as well as the many business benefits of doing so. Finally, you'll learn about the Open Source Program Office (OSPO) and the strategic role it can play in your company's business plan.

Part II: Inbound

[Part II, Inbound \(Using\) FOSS, on page 121](#), first covers the “what even is?” question (hint: it's right in the name) before introducing the concept of FOSS supply chain. It then builds upon this to discuss matters like license compliance, scanning, archiving, auditing, and related complications, such as containers. You'll learn about FOSS sustainability and why a company ignores it at its own peril (not to mention that of the FOSS projects on which it relies). Policies and procedures receive some attention here, along with discussions on the various stakeholders to ensure they're included in all relevant conversations. Leaving them out sets those policies and procedures on a very steep uphill climb to success. Of course the policies you have only matter if people know about and follow them, so you'll also learn a bit about training and internal communications.

Part III: Outbound

[Part III, Outbound \(Contributing to and Releasing\) FOSS, on page 227](#), follows Inbound's lead by getting us all on the same “what even is?” page before diving into a FOSS bucket that holds enormous but often neglected strategic potential for businesses. How can your company contribute to critical FOSS projects while still protecting its intellectual property, including patents? How can you give credit where credit is due but also ensure that your company receives the kudos it deserves for supporting FOSS? What does success look like for your company's FOSS contribution strategy, and what are some of the metrics that can help reveal whether efforts are on track?

Of course outbound FOSS is so much more than contributing to existing projects; it's also releasing new ones into the ecosystem. Why would your company do this, and how does it choose what to release? Once you've chosen what to release, how can your company do so in the most effective and correct

way possible? Releasing FOSS projects is a lot like getting a puppy: it's fun to do, but once you have, then you need to take care of it and ensure it's healthy. What does that mean in this context, how can your company make that happen, and why should it even care—which is to say, what's in it for them? All of this information comes together when considering the topic of business models and how your company can incorporate FOSS into its plans for profitability.

As with most questions posed with respect to FOSS and business, the answers to all these are unique to your individual business and its goals. By the end of this book, you'll understand the variables involved, what that means for your business, and how to answer the question that makes the most strategic sense for your company as well as for the greater FOSS ecosystem.

Suggestions for Tools and Technologies

The tools and technologies available today are not going to be the same tools and technologies available tomorrow. The entire world of software moves at a fast pace, but free and open source software makes the rest look like snails. The public and open nature of FOSS allows it to evolve at incredible speed. It can be difficult to keep up with the latest developments. Thankfully, your company doesn't usually need to stay on the cutting edge of software. It can choose the tools and technologies available at the time it has a problem to solve, whatever those tools may be.

This rapid evolution of software makes it impractical for me to make recommendations for tools or technologies in this book. I could provide suggestions for tools, but that will only frustrate you when you try to use them and discover that they're very out-of-date, are no longer supported, or don't solve your newer problems. Already, recommendations I'd have made at the start of writing the book have been superseded by other tools.

In cases where well-established FOSS options are available, I'll list them. For the most part though, you won't find many "use this!" suggestions. What you *will* find are the concepts and keywords you'll need to do effective searches using your favorite search engine.

One Size Doesn't Fit All

Now that you have an idea of what's in the book, how can you make the most of that information?

First of all, know that there are guidelines and best practices for how to approach FOSS in your company or organization, and you'll find those in

abundance within these pages. However, the practical application of those tips will look different for each company, and sometimes also for different divisions or teams within a single company. That's because each company has its own constraints, needs, and business strategies to consider when approaching this or any business-related decision or policy.

While you may be able to learn some interesting tricks from seeing how other companies approach FOSS, you cannot take their approach, drop it into your organization, and expect it to work any more than you can expect to drop new tooling into your software development life cycle and expect it to work without integration. Sure, it would be a heck of a lot easier if we *could* do those things, but we can't. Instead, you need to take the long way around and figure out the business value your company does or would like to receive from using and/or releasing FOSS, and then build strategies, policies, and procedures to help realize that value.

In lieu of the mythical one-size-fits-all solution for FOSS in business, this book provides not only the background you need to understand the FOSS-in-business problem space but also recommendations for how to gauge the business risks and values that FOSS can bring to your organization and then how to make the most of them. Most chapters first give you a thorough discussion and education about the topic at hand. Take from this what you and your company require for its business and operating models, skipping matters that don't apply and adding others that make sense for you, your people, your FOSS supply chain, and your business needs.

It's tempting to take a shortcut on this path, but it's unlikely to end well. I've seen too many companies end up in a FOSS quagmire that way, throwing millions of dollars away on FOSS projects, products, and strategies that are doomed to failure due to lack of knowledge and preparation. You'll find the price of the book and the time required to read it far easier to manage than a million-dollar failure (and have a considerably higher ROI as well).

Commit to the Effort

While we're talking about avoiding shortcuts, here's another thing you should keep in mind. Like all worthy efforts and strategies, your company's strategic FOSS initiatives will take time to develop, implement, and bear fruit. While a lack of knowledge and preparation certainly scuttles a lot of well-intended FOSS plans, I've seen many well-planned initiatives killed off before they've been given a chance to work. Usually this is due to unrealistic expectations on the parts of the implementors and their superiors. Contrary to common belief, the superpower of free and open source software isn't the software; it's

the people who *create and support* that software who make FOSS the powerful and strategic tool that it is. Like any effort that involves people—such as digital transformation and other efforts involving human culture—changes require time to settle in and take effect.

As you read the chapters that follow, keep this in mind. Certainly your company can gain benefits from FOSS in a quarter or less, but the *really* good stuff is further out on the time scale. Plan accordingly, but most especially set adequate expectations with all of the stakeholders. This concept isn't new to you; strategic efforts take time, and a FOSS initiative is no different.

On the Matter of Jargon

Let's talk for a moment about, well, talking—more specifically, about language and communication and how the former often defeats attempts at the latter.

All too often we put together a string of words that *we* understand without pausing to consider whether the recipient will as well. We assume, and often incorrectly, leading to a failure in basic communication requirements. The prize for the most common type of words to cause miscommunication goes to jargon, which quickly turns even your native language into a foreign one, depending on the audience. Every industry has its own dedicated vocabulary—its own jargon—that its practitioners create and share. This jargon frequently includes a metaphorical alphabet soup of acronyms, each one obscuring a Very Large Idea behind a very small set of letters. This language compression is invaluable for speeding up communication between industry peers. We all do it, and there's usually nothing wrong with it.

The subject of this book requires a lot of jargon. There's no way to escape that. First there's the language of business, with its TCOs and its KPIs; then the language of software and technology, with its APIs and CSPs, its PRs and CI/CDs; next up are the languages of intellectual property law and compliance, with CLAs and SBOMs; and of course there's the language of FOSS itself, with its GPLs and MITs.

It's highly unlikely that you're well-versed in all of these different languages, and that's OK. None of us knows everything. However, if you're going to tackle the matter of FOSS in your company, then you need at the least to become familiar with the many different languages involved. For that reason this book won't avoid jargon, but it *will* go out of its way to explain it, no matter its source language. Readers with a more technical background will learn some business terms, and business folk will learn technical ones. By the time we

reach the end, we'll all be able to communicate much more effectively with each other.

Two resources are included in the book to help with this process. The first is [Appendix 1, Acronyms & Initialisms, on page 385](#). This does no more and no less than unpacking acronyms out to their constituent words, since often all we need is that little nudge to allow us to remember and understand. The second resource is [Appendix 2, Glossary, on page 389](#), which collects and defines jargon from all of the languages involved with the Business of FOSS. You'll find a lot to learn here, and having a glossary at hand means you don't have to memorize a bunch of unfamiliar terms while you're also trying to learn how to make your company more successful through FOSS.

Eschew Obfuscation

Communication is difficult enough on its own. Why make life harder for everyone by using uncommon words where plain, straightforward ones will do? Complex linguistic constructions obscure intent and misrepresent unpretentious hypotheses as euphuistic intellections; which is to say, fancy words make it hard to communicate. It's also a cruel thing to subject your readers to, especially if English isn't their primary language (as is the case for many of you—hello!).

I'm a big fan of plain language, and that's what you'll get in this book. Sometimes sentences may sound a little old-fashioned, since ending a sentence with a preposition is something up with which I will not put. Aside from that, though, I'll do my best to use (never utilize) direct, clear writing.

Links and References

A lot of links and references appear in the chapters that follow, so you'll rarely (if ever) need to raise a *[citation needed]* flag. However, keeping track of all these links so you can find them again can be rather a bother. That's why you'll find them all collected in [Appendix 3, Links from Chapters, on page 405](#), organized by chapter. Because everyone's mental models are different, you may remember a link from [Chapter 1, Lay the Foundation, on page 3](#), while I remember the same link being mentioned in [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#). Therefore you'll find links repeated between chapters in this appendix, and they'll be listed in the order in which they appear in their respective chapters. Hopefully this will make it a lot easier for you to find what you're looking for quickly.

Among the appendices you'll also find [Appendix 5, Other Useful Resources, on page 423](#). Unsurprisingly, this lists other books that I recommend if you'd like to learn more about using, releasing, managing, and contributing to free and open source software. Since it's in a book, this list naturally is static. However, it largely mirrors the list available on <https://fossbooks.com>, which I will update as new recommended books become available.

Join the Online Community for More Resources

The Pragmatic Bookshelf has provided a forum dedicated to this book on the DevTalk platform.² You can join the community of readers in this forum, discussing topics related to the book and helping each other. You can also use the forum to report book errata and make suggestions that might end up in a future edition. Also, there's a dedicated book web page on the Pragmatic Bookshelf website.³

By this point you've figured out that this isn't going to be your standard stuffy business book or inscrutable tech tome. You're here because you have problems to solve and questions to answer, and you don't want to waste time wading through some stifling text to get that done. Aside from the unavoidable need for jargon, this book will keep it casual, readable, and above all *practical*. You and your company have goals to meet. This book will help.

2. <https://devtalk.com/books/business-success-with-open-source/>

3. <https://pragprog.com/titles/vbfoss/business-success-with-open-source/>

Part I

Fundamentals of FOSS

The first part of the book gives you a solid understanding of the concepts you'll need to be successful with open source.

You'll learn what open source and Free Software are, how they differ, and a great deal about intellectual property, copyright, and licensing.

After that, answer the "why does this matter?" question by learning of the risks and benefits of free and open source software (FOSS) to your business.

The part ends with an introduction to the beating heart of open source in your organization: the Open Source Program Office (OSPO).

Lay the Foundation

The world operates on free and open source software (FOSS). From the entertainment software in your car to the database that stores your company's catalog, FOSS underlies it all. This also means that whether you know it or not, your company depends on FOSS. The company phone system may be based on the Asterisk PBX software. The company website and blog may rely on WordPress and its online storefront on WooCommerce. The content production team may use Blender, ffmpeg, ImageMagick, Inkscape, and Krita to develop stunning visuals, while the marketing and community teams are using OBS Studio to create and stream training and demo presentations. Each one of these pieces of software—which may be helping to run your business right now—is free and open source software, and this is only a small handful of the millions of FOSS projects used every day by companies like yours.

Millions of these projects exist because people and companies release more of them every day. Your company doesn't even need to be known for its software, and it may still have released free and open source software to help people solve shared problems. It would be in good company, with media leaders like The New York Times and the BBC; transportation companies like Scania; retailers like Home Depot, Target, and Walmart; financial services companies like Bloomberg and Fannie Mae, and multinational corporations like Samsung, General Electric, Huawei, Siemens, and Bosch all releasing FOSS tools. This even though none of these companies are primarily known for developing software.

Despite the diversity of industries represented by these companies, they all share one thing in common: they all also contribute back to the FOSS projects on which they rely. As you'll learn in this chapter, contributions and support from individuals and companies are the lifeblood of FOSS as well as its

superpower. Millions of individuals collaborate to maintain this software for the good of all its users. Without this community-centered approach, FOSS would fail. Beyond the altruism of supporting the communities that maintain the software on which they rely, the companies listed here and a great many others recognize that the failure of those FOSS tools would put their businesses in jeopardy. They all know that there's a great deal of business value in doing the right thing.

Even with these examples, the thing that most companies share with respect to free and open source software is not realizing the strategic benefits that it brings to the business. In [Chapter 5, Strengthen Your Business Through FOSS, on page 87](#), you'll delve into the details of those benefits, but before we get there it's important you understand exactly what "free and open source" means and why it matters so much.

Why FOSS Matters

Some of what follows is historical background and may not seem relevant to what you're trying to accomplish. Rest assured: your time won't be wasted reading this information. Even a little knowledge about the background of FOSS can help your company avoid a lot of expensive and painful missteps. Let's start with an example of how a lack of preparation can scuttle the best laid business plans.

A Cautionary Tale

Back in 2006, the American company The Home Depot acquired The Home Way, a small chain of retail home improvement stores in China. With the rise of the Chinese middle class and the continual building boom in all of China's major cities, Home Depot was sure that this acquisition would more than pay back its rumored \$100 million USD price tag. Despite that, by 2015 Home Depot had closed all of The Home Way stores and mostly withdrawn from China. What happened?

When they made the acquisition, from the perspective of Home Depot, China was what's known as an "emerging market." These are markets where a company and product have no exposure, or where the business value of a product is not yet recognized by consumers. Because of this, it can be difficult for a company to enter an emerging market. In the case of Home Depot, they overlooked important cultural differences between the American and Chinese markets. In America, many people prefer to perform their own home repairs and improvements. Unfortunately for them, in China it's normal to hire others to do this sort of work for you. The business practices that excelled in America therefore

fell flat in China. The lack of cultural research and understanding cost The Home Depot millions.¹

Entering the Open Source Emerging Market

If your company wishes to be more successful through and with free and open source software, it will need to do so by entering the “emerging market” of FOSS. The massive ecosystem of free and open source software communities and projects exist as a completely foreign culture to that of most companies. Some of these cultures can even be hostile to corporate interests, especially if a company steams into a community with a corporate “business as usual” attitude.

Company participation in FOSS, in whatever form it takes for your company’s business needs, will go more smoothly if you first take the time to understand the culture, language, and expectations of the FOSS communities and projects. These people are more likely to listen and assist if you at least try to speak their language, even if you do stumble over the grammar and vocabulary. Trying to participate without even a basic understanding of the culture of FOSS, how it started, and what that culture means to many of its participants is a sure way for your company’s efforts to fail. By reading this book, you’re going to be more prepared than many of your competitors, setting your business up for success as it enters this FOSS emerging market. What follows is the first step on that journey.

What Is Free Software?

While you may be more familiar with the term *open source*, we’ll start this brief history lesson where it all began: Free Software.

Right out of the gate, it’s necessary to clear up a common misconception. Yes, it’s possible to acquire and use most Free Software (and open source) without any licensing fees; however, the “free” in Free Software doesn’t actually refer to the price. It is, in fact “free as in freedom.” The alternative—“free” relating to price—is known in FOSS circles as “free as in beer,” the obvious parallel being that people like receiving beer for free and so, too, enjoy when it’s software for which they don’t need to open their wallets. This vocabulary collision is a feature of the English language, where the word *free* is unfortunately encumbered by multiple unrelated meanings. Other languages don’t have this problem, and in fact you’ll frequently see the romance language “freedom” variant, *libre*, used

1. <https://www.forbes.com/sites/helenwang/2011/02/10/why-home-depot-struggles-and-ikea-thrives-in-china/?sh=5645ccfb18e4>

along with the English *free* to provide clarification: Free/Libre Open Source Software, also known as F/LOSS or FLOSS. There's no practical difference between the acronyms FOSS and F/LOSS beyond the former being shorter and much easier to type, so with apologies to my romance-language-speaking friends "FOSS" is the one used in this book.

The lack of an up-front price tag is certainly an appealing aspect, but it's the *freedom* element that truly drives the Free Software movement and has since its inception back in 1983. Prior to that, it was common for manufacturers to provide software at no cost, since to them it was the computing hardware that was driving their business models. As a result, computer operators would write, modify, and trade the software. This practice was normal, expected, and endorsed by the hardware manufacturers. It made sense for them, since that software could only run on their hardware and its availability enhanced the hardware feature set, making it more appealing to new buyers. Think of it like a video game that's exclusive to a single platform: if the game is compelling enough, it will help sell a lot more units of that specific video game console.

Eventually the hardware manufacturers got wise to the value that the software brought to their users, and more importantly to their balance sheets. This happened around 1980, when the United States Congress revised U.S. copyright law to declare that software programs could be copyright-protected.² Hardware manufacturers then got wise to the potential revenue stream that creating and selling software could add to their businesses. They were putting a lot of work into creating this software that provided value to end users, and where there's value there's profit. Software development became its own industry distinct from the hardware on which the software ran. By this time, however, people were accustomed not only to receiving the software for free but also to trading it with others. All that ended once the software came with a price tag, and they weren't happy. Not only did it increase their initial costs, the new software industry also meant the end of the user-modified versions that they'd been sharing with each other for years.

One person—frustrated that software operators like him were no longer free to inspect, modify, and share software—took a stand. In 1983 Richard M. Stallman (RMS) announced the launch of the GNU Project, dedicated to the creation of a UNIX-compatible operating system assembled of components that are entirely free to use, modify, and distribute. Two years later RMS

2. <https://en.wikipedia.org/wiki/CONTU>

released the GNU Manifesto, which declares the fundamental beliefs of the GNU Project and launched Free Software as a worldwide movement.

The core of that movement is the Four Freedoms. Those freedoms—which, in standard programming fashion, are zero-indexed and therefore start from zero instead of one—are as follows:

0. The freedom to run the software however you wish and for whatever reason you wish.
1. The freedom to study the software source code and make whatever changes you wish.
2. The freedom to copy and distribute the software (modified or not) however you wish.
3. The freedom to make improvements to the software and then share the improved version however you wish.

According to Free Software advocates, a software license that doesn't guarantee these Four Freedoms to its users cannot be considered “free,” as it limits the rights of its users in some way. To help guarantee these rights and freedoms, RMS, the GNU Project, and the newly formed Free Software Foundation (FSF) created software licenses that leverage the preexisting concepts of copyright. These *copyleft* licenses (a play on the word *copyright*) provide more than the permission to use the software released under them; they also ensure that software can never violate the Four Freedoms. You'll learn more about these licenses in [Reciprocal \(aka Copyleft\) Licenses, on page 44](#).

While many in the FOSS world believe the Four Freedoms are Stallman's greatest invention, in fact his most far-reaching contribution to software is the recognition that copyright can be used in this way, and that careful copyright licensing can enforce software freedom. This invention paved the way for the open source movement that followed.

Note that nowhere in the Four Freedoms does it say a person should not make money from Free Software. As mentioned above, the “free” in Free Software refers to freedom and not to price. From the very beginning, the FSF has encouraged people to use Free Software as the basis for their businesses. Aside from the practical advantage of being able to afford to live, the FSF recognized that a good way to spread software freedom is to show how doing the right thing for users benefits businesses as well. Despite this, you'll still find many who claim to support software freedom yet believe that making money from Free Software is “wrong” or “selling out.”

So Then What Is Open Source Software?

If you ask ten people, “What is open source software?” you may find that you receive ten different answers. The majority of these answers will be wrong. A lot of people think they know what open source software (OSS) is, but typically their definition comes by way of the Telephone Game. They learned it from someone who learned it from someone who learned it from someone... After several retellings, the true message evolves and no longer reflects reality.

Most commonly, you will hear that open source is software “where the programming source code is made freely available,” or some derivative of this statement. Yes, it’s true that the source code (often abbreviated either to *source* or *code*) for open source is available and accessible for anyone, but this is far from the whole picture. The availability of source code is only one of the factors that determines whether software qualifies as open source. For instance, the source code for the Unity real-time development platform is available for anyone to download and inspect but cannot legally be modified or redistributed. Its source is available, but it’s not “open source” software.

The Evolution of Free Software

Let’s return briefly to our friend Free Software. As you can probably tell from the summary above, the driving force behind Free Software is philosophical. For supporters of Free Software, it’s not that people *could* have the freedom to run, study, improve, and distribute the software they use, but that they *must*. Locking software behind restrictive licenses and not providing the source code means that people are unable to learn from that software, modify it to suit their own particular needs, or ensure that the software is secure. It puts the users at a distinct disadvantage to the creators of the software, creating a power imbalance. Free Software supporters believe that by freeing the code we are similarly freeing people, empowering them.

As the Free Software movement grew in the 1990s, it started attracting attention from individuals and corporations. It was obvious that this Free Software thing was bringing a lot to the table, and not all of it philosophical. The Free Software Linux kernel formed the core of an operating system that had successfully overtaken offerings from huge corporations like Sun Microsystems and IBM. In fact, late in the 1990s IBM shifted its strategy to throw support and money behind Linux development, as they recognized the way the operating system winds were starting to blow. Similarly, Netscape released their web browser as Free Software, helping Netscape Navigator go head-to-head with Internet Explorer, the then-predominant web browser from

the software juggernaut Microsoft. These companies saw the business potential of Free Software, and many others were watching and learning.

Some of those who were watching realized that, with a little bit of work, they could make Free Software even more attractive while also iterating on what they'd learned from the first decade or so of the movement. By doing so, they could bring the Four Freedoms to more people while also increasing adoption and release of software that would allow for inspection, modification, and redistribution.

The Dawn of Open Source Software

In 1998 several of these people gathered at an office in the San Francisco Bay Area. The result of this meeting was a rebranding of Free Software as *open source* software. The new name sidestepped the by now (and still) common confusion stemming from the multiple meanings of the word *free* in the English language while also describing the basic nature of the software: its source code was open, allowing anyone to engage in the Four Freedoms should they wish. A crucial part of this rebranding was ensuring that there would be no similar confusion over the word *open*. To avoid this, in true FOSS style, these founders of open source inspected, modified, and redistributed the Debian Free Software Guidelines (DFSG) as the Open Source Definition (OSD). The OSD does exactly what it says on the package: it defines what it means for something to be called “open source,” thereby establishing a standard by which all could operate. To qualify as open source, the licenses used on the project must allow for all ten of the criteria in the OSD:

1. The project must be freely redistributable—even if sold and even if it's part of a larger collection of software.
2. All source code must be available and distributable.
3. Modifications and derived works must be allowed and distributable under the same license terms.
4. If distribution of modified code isn't allowed, it cannot prevent distribution of patch files (snippets of source code that can be applied to include new fixes or functionality) along with the unmodified code.
5. In no way can the license under which the code is distributed discriminate against any person or group. All people must be allowed to use the code on the same terms, even if they're bad people.

6. Similarly, the license also can't single out industries, companies, or other types of undertakings. All groups and ventures must be allowed to use the code on the same terms, even if those groups condone horrible things.
7. The license applies to anyone who receives a copy of the software without needing any additional permissions.
8. The license can't restrict someone from extracting the project or code from a larger collection. If they do extract it, it's available to them under the same license terms as the whole, larger collection.
9. If the software is distributed as part of a larger collection of code, the license can't put any restrictions or requirements on that other code.
10. The license applies to all technology and user interface applications of the software to which it's applied.

A software license that doesn't meet each and every one of these criteria violates the Open Source Definition and—literally by definition—is not open source.

These people created a new nonprofit organization, the Open Source Initiative (OSI), to act as a warden over the OSD and to promote the use and release of open source software across the world. OSI ensures that the license of software you use does, in fact, comply with the Open Source Definition and therefore is truly open source. As you look at the ever-growing ecosystem of OSS, including many components that are mission-critical for your own business, it's evident how large and important of a job the OSI has. Thankfully, from the outset the organization recognized that reviewing and certifying every single project claiming to be “open source” was not going to scale. Even back in 1998 FOSS projects were being released at an impressive rate, a rate that has grown dramatically every year since. Rather than taking on the impossible burden of certifying projects, the OSI instead reviews and certifies *licenses*.

When a person or organization has a software license that they believe obeys the Open Source Definition, they can submit the license to OSI for review and potential approval. A team of FOSS and legal experts then compare the license to the OSD. If software is released under the proposed license, will it guarantee each of the ten criteria? Is the license well formed (and ideally professionally drafted), without ambiguity as to the rights and responsibilities of both the license holder and receiver? If so, OSI can certify the license as compliant with the Open Source Definition. From there on out, anyone who uses software that's released under that license can be assured that the software is indeed open source.

For instance, consider the MIT License. Approved by OSI, this license is short, sweet, and to the point:

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice (including the next paragraph) shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

You can see that the license explicitly declares that everyone who uses software released under this license has the freedom to run, study, redistribute, and improve it. The license contains no clauses limiting who can use the software or in what way. If the OSD is a checklist, then the MIT License ticks every box and easily passes the open source test. It’s one of the most popular OSI-approved licenses in use today.

By putting the focus on the license instead of the project, OSI has created a quick and easy way for companies and individuals to release and use FOSS projects without needing to consult a lawyer. Of course, companies in particular should still always consult a lawyer when entering into a contractual-type arrangement, such as the one embodied in a software license. However, having a standard set of certified licenses makes this job considerably easier for them. Faced with an OSI-approved license, the lawyer, user, or company will broadly know what to expect in the rights and responsibilities they have under that license. It makes everyone’s job considerably easier. You’ll learn all about FOSS licensing in [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#).

The Difference Between Open Source and Free Software

Open source grew from the seed of Free Software, so why keep talking about Free Software at all? When you have an oak tree right in front of you, you don’t usually keep talking about the acorn. While open source currently

receives the majority of the mind share, Free Software is still very much alive and well, as evinced by the success of projects such as Linux, Blender, Firefox, and so many more. Open source shares the same genetic base code as Free Software and the two complement each other well, so much so that from a user perspective there often appears to be little practical difference between the two of them. They can download, install, and use the software; they can inspect, modify, and redistribute the code. In fact, the Four Freedoms are included in the Open Source Definition, making nearly all Free Software, by definition, open source software, but differences remain.

Those differences are primarily cultural and influence some practical matters. Supporters of Free Software maintain that the goals of their movement—user and software freedom—require that the software always remain free/libre, including any modifications that might be redistributed. To accomplish this, Free Software is released under *reciprocal* (also known as *copyleft*) licenses that require that any modifications be released under licenses that will similarly ensure user freedoms (you’ll learn more about this in [Reciprocal \(aka Copyleft\) Licenses, on page 44](#)). Once a project is Free Software, by the terms of the license it must be Free Software forever. This is an over-simplification of a complex licensing matter, but it’s a *useful* over-simplification. Use it as a rubric when considering your company’s use of FOSS released under reciprocal licenses. From a cultural perspective, Free Software advocates believe strongly in the Four Freedoms and can be fairly dogmatic about this. When working with the community of a Free Software project, keep this belief in mind and respect it in your community engagement.

Open source, on the other hand, is more permissive about several things. Much open source software can be modified and redistributed under completely different licenses (an uncommon and often complex process, but allowed under the license), including in some cases under a license that closes the source code entirely. In that case, the software would cease to be open source and would become proprietary. Such a thing would be impossible in Free Software.

But again, in practice, you may not notice much difference between the two, and they are highly complementary. This is why you see them combined in the FOSS (free and open source software) acronym: for the most part they can be treated as a single entity in conversation. That’s how they’ll be treated in this book, but never forget that they are different things. That knowledge will come in handy when engaging with the FOSS communities your company relies on or builds and manages themselves. Communities that identify as

Free Software will have a different set of cultural expectations than those that identify as open source.

Community and Contributions

Before we go much further, here are a couple of concepts you need to learn about: community and contributions. Each of these is vital to FOSS, to the point that the entire free and open source software ecosystem would collapse if either were not there. In ecologist terms, this makes them keystones for the ecosystem.

You're going to see the words *community* and *contribution* a lot throughout this book, so we should take a moment to make sure we're all on the same page as far as what these two keystone concepts actually are.

Community

A community is a self-organized and self-identified collection of people sharing a concern or interest and, aside from the Four Freedoms and the Open Source Definition, it may be the most important thing in FOSS. Communities can be large, such as the overall community of FOSS users, maintainers, and contributors. This community numbers in the millions. They also can be small, such as the handful of people who come together around a small niche project, such as the *ih* software for creating cross-stitch patterns.³

FOSS puts a great deal of stock into the concept of community. For a project to be considered healthy, it must be driven by and for the community. When making decisions, options that favor or strengthen the community typically take precedence. Projects that don't operate in an open manner—so the community can witness and provide feedback—or are operated primarily by a single company or organization, fail the community test. These projects may meet all the criteria of the Open Source Definition, but the lack of community engagement makes them appear less “open source-y” to outside observers. This can limit adoption of a project, as well as its growth and sustainability.

Some projects intentionally have no community. The project creator, for whatever reason, is done with the software. They don't need it anymore and would like others to benefit from it, but they have no intention of maintaining the software, answering support questions, or sparing any thoughts for the project at all. They release it, then metaphorically dust off their hands and walk away. In FOSS, this action is known as “throwing it over the wall” and is a

3. <https://github.com/glasnt/ih>

perfectly legitimate approach to releasing a project—as long as the creator sets up appropriate expectations in the project repository and documentation.

Except for the case of throwing a project over the wall, the default expectation is that any project released as FOSS will act in a community-oriented fashion. Therefore you'll read a great deal about community throughout the course of this book. This will help you keep community front and center whenever using, releasing, or contributing to FOSS. You'll learn more about this in [Chapter 20, Releasing FOSS Projects, on page 317](#).

Contributions

The open nature of FOSS enables the formation of communities and so much more. Not only can anyone use free and open source software for any purpose whatsoever, anyone can also improve it. This happens by way of *contributions*, which allow FOSS to innovate and evolve more rapidly in the open than proprietary software developed by a single team in secret.

FOSS contributions take many forms. Of course you'll see the typical contribution of software code but also security scans, project management, documentation writing, marketing, testing, bug triage, community management, and so many more. Any of the functions and roles that are a part of the creation or maintenance of any software are also necessary for FOSS projects and therefore can be contributed. Project maintainers, however, aren't obligated to accept all contributions. For instance, a maintainer may choose to reject contributions from organizations or individuals with whom they disagree on ethical or political grounds or because they cannot or will not accept the maintenance burden of the contribution. The maintainer may also have thrown their project over the wall, opting to release the code but not to support or improve it. These situations are exceptions to the rule, and typically maintainers are happy to accept contributions from the project users and community.

You'll learn some things about contributions in [Chapter 17, Master Contribution Fundamentals, on page 263](#), but for the complete picture of what and how to contribute, check out *Forge Your Future with Open Source*.⁴ It's the only book entirely about how to make contributions to free and open source software projects and communities.

The Importance of Standards

You know the appliance you buy will have a plug and internal wiring that'll work with the electrical sockets in your home. You know the Bluetooth

4. <https://fossforge.com>

headphones you bought will work with your smartphone. You know the kilogram of rice you buy at the store weighs exactly the same as the kilogram of rice you bought at a different store last week. You know these things because they're based on standard specifications (*standards*) that have been agreed upon by governments and practitioners all over the world.

Standards and Business

Without standards, your company would have a difficult time doing business. Let's say you're in the grain industry, packaging and distributing the aforementioned rice to stores for consumers to buy and take home to feed their families. Numerous standards ensure that you're able to operate without having to second-guess everything you buy and sell. Standards are in place for the quality of rice, whether it's grown using organic methods, and precisely what mass of rice constitutes a kilogram. Organizations exist solely to create these standards, and others are dedicated to reviewing and certifying that the standards are being followed. Both types of organization do this work so your company doesn't have to and is therefore freed up to literally go about its business.

Now what would happen if, somewhere in your rice company's supply chain, there were a deviation from these standards? Your company could be paying a premium for wholesale organic rice but receiving cheaper rice grown using conventional methods or, even worse, it may be paying for a certain number of kilograms of rice but receiving the same number but in pounds instead (a quantity less than half what you were expecting) because the wholesaler is working under a different definition of "kilogram" than you are. Imagine the chaos that would ensue in your business if your company couldn't rely on standards. Standards enable businesses to operate with smooth communication by creating a shared and trusted language and understanding.

Standards and Open Source

By having a single, accepted specification for what's required for a software license to be open source, what the OSI has created and maintains with the OSD is a *standard*, with the OSI playing the role of standards body. Like other standards, this one is vital for the operations of any business that uses or creates software (which is to say nearly all of them). When your company receives software that bears the label "open source," you know that while there may be some responsibilities to be aware of in the license, you now have a firm grounding in generally what to expect from that software. You know that the license won't prevent your company from using the software in certain

situations, that if needed your company can modify the software to support its business needs, and that it can even sell the software if that's what makes sense for its business model. It knows these things because the software is provided under a license that the OSI has certified obeys all ten criteria in the Open Source Definition.

Imagine if your company received and starting using software that claimed to be “open source” but was operating under a different definition. Perhaps the provider of the software has strong objections to rice, so they modify an otherwise OSI-approved license to include the stipulation that the software cannot be used in any application that supports or promotes the use or promulgation of any sort of rice or rice products. If the software is ever used for that, the perpetrator must pay the software provider 50 percent of their income from the past year. Your grain company, believing the software must be obeying the OSD because it's labeled “open source,” proceeds to use the software to build the logic for its grain-bagging machine. If that machine is used to bag rice, your company has violated the license and now owes a great deal of money.

Unfortunately this sort of thing is happening more often—software labeled as “open source” but released under licenses that violate the OSD in one or several ways. This *fauxpen source* (faux open source) practice is eroding trust in the open source standard and making it considerably more difficult for businesses to operate. As with all standards, the appropriate response to ecosystem evolution is not to ignore the standard in favor of your own solution—the technique of these fauxpen source companies—but instead to similarly evolve the standard to be more suitable to the environment. Your mobile phone, for instance, likely uses a USB 3.1 connection with a Type C plug. The Universal Serial Bus and associated plug standards have both evolved several times over the years via collaboration of industry members in the USB Implementers Forum. Similarly, there's room for evolution in the standard of the OSD but only if it's also approached in a collaborative way under the auspices of the warden of the standard, the Open Source Initiative.

Now you have an understanding of the meaning and importance of free and open source software to your business and how it came to play the pivotal role it has today. The ability to use, modify, and redistribute FOSS allows your company to move more quickly,⁵ but how is it you're able to use that software at all? The answer lies in *intellectual property law*, and it's the foundation on which all FOSS is built.

5. <https://www.redhat.com/en/resources/state-of-enterprise-open-source-report-2022>

Make the Intangible Profitable with Intellectual Property

As you learned in the last chapter, Stallman used existing copyright practices and laws to create *copyleft* and Free Software. This then directly led to the Open Source Definition, the Open Source Initiative, and open source software. Copyright still underpins it all, as does copyright's bosom buddy *licensing*. In fact, the concept of licensing is so tightly entwined with FOSS that it's often the first thing people consider when they think about free and open source software. It's such an important concept that it gets its own chapter, [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#), which will deep dive into the concept of licensing, the many different flavors of licenses, and how they affect and interact with FOSS.

IP Is the Foundation of FOSS

Copyright and licensing are both foundations of FOSS, and they're both also important elements of the larger world of *intellectual property* (IP). It wouldn't be a stretch to say that copyright and licensing aren't the actual foundation of FOSS, IP is. IP is not only the basis of FOSS, it's also a critical element in nearly every business. Furthermore, it's something about which most business managers have surprisingly little knowledge. We'll fix that in this chapter since you'll need that knowledge to make informed strategic decisions in your company, and not just about FOSS. By the end of this chapter you'll have enough of a grounding in intellectual property concepts not only to understand the remainder of the book but also to be able to spot potential IP opportunities or risks within your own company. A little IP knowledge can go a long way.

Lawyers Are Your Friends

Before we dive too deeply into legal topics, it's time to learn a couple of acronyms: IANAL and TINLA.

IANAL is short for “I am not a lawyer,” TINLA for “this is not legal advice.” Both arise frequently in conversations about free and open source software and are almost always followed by “but...” and then an explanation of some sort. Due to the organic and community-driven growth of FOSS, many laypeople have many good insights about the intricacies of FOSS-related legal matters. Many also believe they know what they're talking about but have more self-confidence than valid knowledge. Regardless, it's become standard practice to use these acronyms to help ensure one's audience doesn't have the expectation of having received advice from a certified legal professional.

Lawyers will sometimes use TINLA to let people know that while they *are* a legal professional, they are not currently speaking in that capacity, as doing so brings with it certain professional and ethical obligations and duties. They'll sometimes follow it up with IANYL (I am not your lawyer) to make it very clear that there's no lawyer-client relationship implied by their words, further avoiding confusion about whether the lawyer is providing official legal advice (something that typically and justifiably comes with a price tag and possibly liability).

As for this book, both IANAL and TINLA apply. I'm a highly experienced and knowledgeable business strategy professional with over thirty years in FOSS and its communities, and the advice I provide is based upon that. I have an abundance of experience and knowledge, not formal legal training. However, that experience and knowledge is comprehensive and laser-focused on what you need: understanding how FOSS works so that your business is more successful by being a good FOSS citizen.

So much of business operations hinges upon intellectual property and so many business models rely upon it, that business managers should have a grounding in basic IP concepts such as these. That knowledge (or lack of it) can make or break a company. Even with this grounding, you'll still need ready access to IP legal counsel but less than otherwise. It's like getting injured: you've learned through experience that some cuts require only a bandage while others need someone with medical training to stitch them up. The same holds true for IP law: sometimes all you need is a bandage, but other times you'll need professional help. We'll return to various IP topics throughout the course of the book; for instance, the choice of license when releasing or using FOSS components can act as a powerful lever in the business strategy machine. By the time you reach

the final chapter you'll have a better sense for when you need an IP bandage or stitches, metaphorically speaking. However, whenever there's even the slightest question or doubt *always consult with a legal professional*.

Introducing Intellectual Property

What exactly is intellectual property? Here's the definition according to the World Intellectual Property Organization (WIPO):¹

Intellectual property (IP) refers to creations of the mind, such as inventions, literary and artistic works, designs, and symbols, names and images used in commerce.

That is to say, the intangible products of intellect are all intellectual property. As property, these things have owners and, like property, they're protected by law. Intellectual property is a large and complex practice of law. It covers topics such as trademarks and other trade dress, trade secrets, patents, and copyright with its frequent companion licensing. As you'll see throughout the book, FOSS can free companies and individuals from some IP concerns but not all of them. Each one of these IP topics is complicated, and each has some very sharp edge cases. Because of this, it's important that your company either employ or have ready access to professional IP legal counsel. The cost of a one-hour meeting to verify or correct assumptions is money well spent when it can save the company thousands or millions from IP missteps.

Trademarks

Trademarks—a blanket term that can refer to trademarks, service marks, trade dress, and related product and service characteristics—are typically entwined with commercial or business endeavors. By distinguishing the goods and services of one company or party from those of others, trademarks help to establish, preserve, and maintain a brand. Of all the forms of intellectual property, trademarks are historically one of the earliest recognized and are protected by law. Trademarks—literally the unique identifying mark of the tradesperson who created the product—have been found on items that are thousands of years old. Trademarks allow consumers to judge whether this type of car is reliable, that type of tea is the one you prefer, or those clothes are assembled in poor factory conditions, and enable them to select between these different brand names quickly and easily. From a business perspective, trademark protections allow a company to safeguard its reputation from competitors and charlatans who could otherwise try to capitalize on the hard work someone else put into building a strong brand.

1. <https://www.wipo.int/portal/en/index.html>

As you’ve probably already guessed, these trademark protections are a matter of law. While the details of protections vary, just about every jurisdiction has some sort of trademark law on the books. This is important, because trademarks have a geographic component: a company can only register and enforce a trademark in locales where that trademark either is or imminently will be used. The actual process of receiving protections for and registering a trademark will also differ by jurisdiction. For instance, in jurisdictions based on common law, such as England and the United States, you can acquire some form of protections for a trademark simply by using it to do business, with no further registration required. Those protections may be minimal compared with what you get by registering the trademark, but they’re adequate protection for some (typically very small) businesses.

Distinctiveness is important for trademarks. To qualify for protection under the law, the trademark must not be some general descriptive term or mark. It must be as unmistakable (distinct) as possible, but it doesn’t need to be unique throughout the world. Like I said, these things have a geographic component. You can open a tea shop named “In Hot Water” in Accra and then register the name as a trademark in Ghana, and this would in no way impact someone else registering the same tea shop name in Sydney—unless you operate a chain of tea shops and have already opened one and registered the name in Australia. The two service areas are *distinct*. This distinctiveness also doesn’t cross the lines of product or service area. Someone else in Ghana could register a trademark for a laundromat under the same “In Hot Water” name. Because the tea product is *distinct* from the laundromat service, there’s no consumer confusion between the two brands and the trademarks coexist peacefully. These three forms of distinctiveness—in mark, in jurisdiction, and in business area and operations—can determine the protections available for a trademark.

The matter of trademarks doesn’t come up too frequently in FOSS, but when it does it’s important. This can happen when a company is looking to release its own FOSS project or advertise that it uses an existing project, but also when a community around a FOSS project wishes to protect its identity via trademark. For instance, all projects under the Apache Software Foundation are protected by its trademark policy, which dictates how and under what circumstances external bodies may use a project’s name, logo, and other trademarked assets. You’ll learn more about how trademark comes into play in [Chapter 20, Releasing FOSS Projects, on page 317](#).

Patents

Patents protect inventions: ideas for products or processes that either are a new way to do something or are a technological advancement on an existing method. Taking out a patent on an invention gives the creator the right to profit from their idea for 20 years (though, as you likely expect, the length of time can vary by jurisdiction and type of patent). The inventor can produce and distribute the invention themselves, license others to do it for them, or sell the patent rights to someone else entirely. Regardless, until the patent expires, only the entity that holds it is allowed to use or produce that invention.

For trademarks and, as you'll learn shortly, copyrights, there's often no requirement to register prior to claiming the rights to which you're entitled for them. For countries such as the United States and Germany that acknowledge common-law trademarks, you get at least limited trademark rights automatically upon use or creation. Not so with patents. The only way for an inventor to receive a patent for their creation is to apply for it, and even then the patent isn't guaranteed.

To qualify as a patentable invention, it must be unique and nonobvious, which is to say no one must have created it before (*prior art*), and someone with an "ordinary" level of skill in the industry where the invention applies wouldn't think that the invention is obvious and not at all innovative. If the invention passes these criteria, then it must be publicly disclosed in the patent application. That disclosure must be complete and clear enough that someone with an "ordinary" level of skill in that industry could reproduce it. The application is public, and it only applies to that one jurisdiction where you filed the patent application. To receive a patent for the same thing elsewhere in the world, you'll need to apply for the patent again in other jurisdictions. As you can imagine, this can get tedious and expensive, but that's how it works.

For example, consider the case of Bees4Less: The Pollination Station. This (entirely fictitious) company provides wholesale and direct-to-consumer bees and beekeeping products to everyone from hobbyists to large agricultural concerns. Their Bee Boffins invented a new method for cooling hives during the increasingly hot summers. They were able to qualify for and receive a patent in the jurisdiction where they do most of their business, the United States. If they wish to expand into the Canadian market, they'll need to apply for and receive a patent from the Canadian Intellectual Property Office.

Should someone take that patent application and reproduce the Bees4Less invention while they still have a patent on it, Bees4Less, as the holder of the patent, can sue them for patent infringement and/or ask them to pay fees to license the hive-cooling patent. The license would give them the right to use the invention in the ways detailed in the license itself. This is one of the ways patents help inventors; they ensure that the inventor can profit from their work. This assumes, of course, that the patent holder can afford the costs of a possibly lengthy legal battle for infringement. For many small businesses, this isn't possible. Despite that, having a patent is rather like having a big guard dog: even if it never barks, it still provides a fair amount of protection and can dissuade others from infringing on the patent in the first place.

Patent Clauses

In some jurisdictions, software and/or business processes can be patented. These are the primary ways patents intersect with FOSS: code is released under an open source license, but that code includes a novel approach to accomplishing a task. The creator may pursue a patent for that software, but then how will that affect those who use the open source project? Wouldn't they be infringing on the patent if they were to use the software? Possibly, yes. For the most part, it all depends on the license applied to the project.

You'll learn all about licenses in [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#), but for now it's enough to say that some licenses include clauses that clearly state that if the software includes any patented bits, then anyone who uses, modifies, or redistributes the software also has a license to use the patents that are in the work. The most popular license with an explicit patent clause is Apache-2.0. Section 3 of this license states:

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

This patent clause makes the Apache 2.0 License popular with businesses, but other FOSS licenses contain either explicit or implied patent clauses of their own. For instance, some IP lawyers convincingly argue that the MIT License contains an implied patent grant.²

Regardless of the license applied to a FOSS project, your company should tread carefully where software patents are concerned. Patents are an intensely complicated area of IP law, and software patents add a whole extra layer of complication on top of that. First of all, it's difficult to determine whether a software invention qualifies for patent protection, but added to that is the problem that many in the software industry, and especially in the FOSS ecosystem, believe that software *should not* be patented at all. This belief can add tension or conflict when a FOSS community discovers that a company that's contributing to the project is also taking out patents over their contributed code. It can be OK to do this (and can be a strong business strategy), but it must be approached with care and empathy when a FOSS community is involved.

Everyone into the Patent Pool

Of course, with any tool comes the possibility for misuse, and IP is no exception. For instance, unsavory individuals and corporations will buy patent portfolios, typically of small or failing businesses, and then use threat of patent infringement lawsuits to extort licensing fees from others. These entities—commonly known as *patent trolls*—are motivated purely by greed and have been known to operate in ways that, while technically legal, are more suited to the back rooms of organized crime than the boardroom.

To help combat this practice, many companies join *patent pools*. All members of the pool agree that other members are allowed to use the patents they place in the pool without fear of legal or financial action. Through this collective action of the members, they form a coalition that holds off patent trolls. The largest patent pool in the world is the Open Invention Network (OIN), created to pool and protect patented innovations present in the Linux kernel and related FOSS technologies. OIN is unique among patent pools because not only does it defensively protect FOSS components from patent complications, but it also regularly redefines and expands which components are covered by its protections. In this way, OIN licensees can see their benefits expand along with the free and open source software ecosystem.

2. <https://opensource.com/article/18/3/patent-grant-mit-license>

Copyright

All the topics of intellectual property will come up at some point or other in FOSS, but copyright is one that's so fundamental to all of free and open source software that it deserves a detailed explanation all on its own. But what is it? Here's according to WIPO:³

Copyright relates to literary and artistic creations, such as books, music, paintings and sculptures, films and technology-based works (such as computer programs and electronic databases)...Copyright protects the owner of the exclusive property rights against those who copy or otherwise take and use the particular form in which the original work was expressed.

The United States Copyright Office further clarifies that copyright covers “original works of authorship fixed in a tangible medium of expression.”⁴ Copyright covers both published and unpublished works.”

Summed up and translated for nonlawyers like you and me, copyright is exactly what it says on the tin: having the right to control the copying, use, repurposing, or distribution of a work. A *work* is anything that you create regardless of whether it qualifies for copyright protection.

While intellectual property concepts of patents and trademarks protect *ideas*, copyrights protect the *expression* of an idea. If you've expressed an original idea in some fixed or permanent state, then it's probably protected by copyright. Three important criteria in that last sentence bear a little more scrutiny: *expressed*, *original*, and *fixed*.

Expressed, Original, and Fixed

Let's say you go to sleep tonight and dream of the most beautiful piece of music the world has ever known. No one has ever heard music like this before. Immediately upon waking you write down the song so it doesn't dissolve, as dreams often do (because, of course, in fictional examples we all have the skills required to notate music and keep pads of musical staff paper at our bedsides). By putting that new music into writing, you've *expressed* an *original* idea in a *fixed* form. You've also instantly acquired copyright over it.

When you wake, that music is no more than an uncopyrightable idea because it exists purely in your brain and nowhere else. That the idea is original doesn't matter because it doesn't really “exist” as such. However, the moment you document that song in some form, you *do* have copyright over it, and

3. https://www.wipo.int/edocs/pubdocs/en/wipo_pub_909_2016.pdf

4. <https://www.copyright.gov/help/faq/faq-general.html>

instantly, as long as that documentation is a permanent form. Writing it on the beach at low tide or as sky writing on a windy day is unlikely to qualify (but an IP lawyer may know of a clever way to make it work). The form of fixed expression doesn't have to be formalized in any way besides continuing to exist in some reasonable way. The beach or sky writing doesn't have the permanence required for copyright, but sketching out the melody on the back of a paper napkin or singing it into a voice memo on your smartphone does.

The Low Bar for Originality

The preceding example shows how easy it is for a work to qualify for copyright protection: it has to exist in a fixed form. The criterion of originality is also one that's interpreted fairly liberally. Generally, it's not too hard for a work to qualify as original and be copyrightable. Unlike with patents, copyrightable works don't need to be *novel* and entirely new, unique, imaginative, or anything of that sort. The bar for originality and creativity is set very low for copyrightable works, and you'll find that in practice there aren't a lot of things that fail to meet these criteria.

What Can't Be Copyrighted?

Some categories of things can't be copyrighted no matter how original they may be. For instance, you've already learned that ideas don't fall under copyright, only the *expression* of an idea can. Some of those ideas are covered by other IP law, so they can't be copyrighted, such as names and logos (trademark law) and inventions (patent law). Certain types of identifiers—such as internet domain names and the titles of books and movies—don't fall under copyright. This is how directors are able to keep making films and TV episodes with the title *The Awakening* (you'll find more than thirty of these—so far). Facts and common knowledge data cannot be copyrighted, so you're in no danger of copyright infringement if you perform a dramatic reading of the periodic table of the elements.

Most people don't know that practical things can't be copyrighted. This includes useful objects like appliances but also clothing and fashion. There's nothing illegal about copying the design of the newest Gucci purse and selling your knockoff product. But if that purse is covered in Gucci logos and you copy *those* in your design, then you're infringing on the Gucci trademark and will probably be sued when they catch you.

Also falling into the practical-so-not-copyrightable bucket are lists of ingredients or components, such as formulae or recipes. If you've ever wondered why so many food bloggers include pages of text before their recipes, now you

know: while their recipes aren't copyrightable, those pages of text *are* and form a sort of rudimentary IP-protective barrier around the recipe.

Finally, works created by or for certain state governments (such as the United States Federal Government) cannot be copyrighted and are in the public domain. This practice varies by jurisdiction, so do check before just assuming that any government works are free for the taking.

Software and Copyright: It's Complicated

As you've probably already guessed, software is one of those things that *does* fall under copyright. The source code and related files are the expression of the ideas of the creators. Therefore, whoever writes the code (or the entity for which they're writing it) has copyright over their creation. But, as you've probably also guessed, the relationship between software and copyright falls firmly in the It's Complicated camp.

According to copyright law, software counts as a literary work.⁵ It's the written expression of an idea. Remember, though: only the expression falls under copyright, not the idea itself. Therefore, our friends at Bees4Less could write a piece of software that monitors the temperature inside a hive, and they would have copyright over that software. However, I could see their software, think, "Gee, I could really use a way to monitor temperatures!" and then implement the idea myself. Unless they somehow managed to receive a patent on temperature monitors, I would be entirely within my rights to create this new expression of an existing idea. They could sue me for copyright infringement anyway, but they'd need to prove that my program copies theirs. Some of the criteria used to prove this include the structure and organization of the original source code, so even if I were to use a different programming language, if I use the same code structure (even coincidentally), then it can appear as if I copied their program. In the eyes of the court it may look like my program is an *adaptation* or a *translation* of theirs, which would be a violation of copyright. See? Complicated.

Copyrights and Algorithms

Adding to that complication is the fact that while the source code for a program is copyrightable, the *algorithms* that the code implements are not.⁶ These fall into that practical-so-not-copyrightable bucket I mentioned previously, where they hang out with scientific and mathematical formulae. In the process of

5. <https://www.wipo.int/copyright/en/activities/software.html>

6. <https://www.copyright.gov/circs/circ31.pdf>

writing their temperature monitoring software, Bees4Less may implement an algorithm that calculates the ideal temperature based upon the size and population of the hive, the ambient temperature of the surrounding air, the current height above sea level, and the hardiness of the species of bee. Because this variable-based calculation they came up with is a formula, I'm free to implement it in my own hive temperature monitor—unless they patent it. Patents exist to claim ownership and provide protection for inventions and original ideas, and novel algorithms can qualify as such. We'll cover patents more in [Chapter 20, Releasing FOSS Projects, on page 317](#).

Shared-Authorship Means More Copyrights Are Involved

Just when you think we've reached the end of the complications for copyright and software, the matter of shared authorship swaggers into the conversation. This complication weighs fairly heavily on FOSS projects, since nearly all of them are the work of multiple programmers and contributors. Generally speaking, each of those programmers and contributors retains copyright over their specific contribution to the software. Much like individual polyps come together to form a single beautiful tropical coral, individual copyrighted contributions form a single FOSS project. While usually the project can now be treated as a whole—just as we typically speak of the coral as a single entity—sometimes the rights of the individual copyright holders come into play. For example, relicensing of a project can require the input of all copyright holders since the new license may change how their work can be used. Relatedly, should someone fail to comply with the chosen license of a project, any of the contributors whose copyright is involved in the infringement could take legal action against the violator.

Nontriviality

But wait, there's more! There's also the issue of originality and nontrivial derivatives in contributions to free and open source software projects. You see, while typically each contributor to a project automatically receives copyright over their contribution, this in practice only happens when that contribution is original and nontrivial. Trivial contributions don't qualify, because copyright covers *expressions of ideas*. Expressions of obvious things (trivia) need not apply.

The legal definition of *nontrivial* is vague and feels like an “I'll know it when I see it” sort of situation. If someone contributes a typo fix to a FOSS project, is that contribution trivial? Possibly. If so, it doesn't qualify for copyright protections. But what if that typo fix corrects a build error or a flaw in the

functionality of the software? Is it still a trivial contribution? That's a gray area.⁷ These are matters for the courts to decide, if and when it comes to that. You can see how helpful it can be to have IP legal counsel available for your company to help you navigate this sort of mind-bending question.

You Wouldn't Copyright an API

Different pieces of software interact by way of Application Programming Interfaces (APIs). A program exposes (and hopefully documents) its API, essentially publicly declaring the grammar, syntax, and vocabulary another program must use if it wishes to communicate. Since the entirety of software development and the internet relies on one piece of software being able to communicate and interact with others, APIs are critical to the smooth operation of, well, of just about anything with software in it.

In 2010, Oracle sued Google for, among other things, infringing on Oracle's copyright by copying APIs from the Java programming language (the rights to which Oracle had gained by acquiring Sun Microsystems) for Google's own Android operating system. This lawsuit rocked the software world. If Oracle won its lawsuit, it would mean that APIs fall under copyright protections in the United States and nearly all of the interoperability enabled by APIs would likely become illegal. Imagine you know the language you should use to speak with someone else, but actually doing so could earn you a lengthy and expensive legal battle. All communication would stop. This was the worst-case scenario the software industry pictured should the suit be found in favor of Oracle.

The case took ten years and several appeals to run its course, eventually being decided by the Supreme Court of the United States (SCOTUS). In April of 2021, SCOTUS released its ruling that Google *had not* violated Oracle's copyright by reimplementing Java APIs. Instead, the court found that Google's actions were *fair use* (you'll learn more about this in [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#)). The software world hailed this as a major victory as well as a bullet dodged. Software interoperability was saved! Or was it?

The fundamental question of the case was "Can APIs be copyrighted?" In finding that Google's reimplementation of the APIs was fair use, SCOTUS entirely sidestepped the question of copyrightability. The question remains unanswered, so it's still possible that another API copyright case may yet come before a lower court. However, considering how much time, effort, and money these two juggernauts of the software industry put into the Google vs. Oracle case, only to reach an indeterminant answer to the question, most experts agree that we're unlikely to see such a case any time soon.

So, for now at least, software developers are free to use (and reimplement) the APIs of other pieces of software to build the complex functionality required to operate modern business.

7. <https://softwarefreedom.org/resources/2007/originality-requirements.html>

Getting Copyright

With all those complications, the process of actually acquiring the copyright over a work must be a convoluted mess, right? Nope! The moment an idea is expressed in fixed form, someone instantly owns the copyright to it with no paperwork needed. This is called the *principle of automatic protection*, and it exists thanks to something called the Berne Convention for the Protection of Literary and Artistic Works, more commonly and simply known as the Berne Convention.⁸ Currently, 179 governments have signed this international agreement, effectively meaning the entire world shares the same understanding of the fundamentals of copyright.

However, that doesn't mean there's never any need to register copyright ownership over a work. The requirements for this vary by jurisdiction. For instance, in the United States copyright registration is required prior to bringing legal action for infringement on that copyright. Registering copyright prior to that also can have benefits to the copyright holder should they bring and win a suit for someone violating their copyright. As with all matters involving intellectual property and your business, always check with legal counsel to see whether it makes sense for your company to register copyright over its works.

Who Gets the Copyright?

You may have noticed that I said the moment an idea is expressed in fixed form *someone* instantly owns copyright to it. You'd be correct to wonder exactly who that someone is, since the answer has direct bearing on your business. While the basic answer is "whoever created the work has copyright," it's not always that simple. Were it always the case that the person who created it owns the copyright, then your company would own *no* copyrights whatsoever. Instead, each employee of your company would have copyright over the works that they created, and the company would be unable to guide, change, release, or control any of it unless the employee gave them license to. As you know, that's not actually the case, and your company does own the copyright for all the things its employees create for it. This is thanks to *work for hire*, which says that the answer to the question "who owns the copyright?" is either the person who created the work *or the entity that paid them to create it*.

Work for Hire and Its Effect on Copyright

Work made for hire (commonly known as work for hire) is a statutory concept in the United States and is related to international concepts like *corporate*

8. <https://www.wipo.int/treaties/en/ip/berne/>

authorship and *copyright assignment agreements*. The details vary per jurisdiction (including per state in the United States), but the basic idea is if a *legal person* pays for someone to create a work for them, that legal person receives certain rights over the copyright of that work. A legal person is anyone *or anything* that can enter into contracts, own property, and other legal stuff like that. This means your company most likely qualifies as a legal person. As such, the copyright for works created for the company is—in many jurisdictions—owned by the company. Whether this is actually the case will depend upon that jurisdiction, the laws in force there, the contracts and/or agreements signed by the employees when the company hired them, and other factors.

As is common in IP law, there are edge cases in work for hire, and some of those edges can be very sharp. This is especially true when hiring employees outside of the state or country where your company is based. I won't go into the gory details of economic versus moral rights or dualistic versus monistic legal systems, since I like you and won't put you through that. The bottom line is that while a company can reasonably assume work for hire holds true in the United States, the same can't be said for other countries. This is yet another of the many examples why your company should always consult legal counsel when dealing with intellectual property and employment matters.

Practically speaking though, all this means your company likely owns the copyright to everything its employees create on the company's behalf. Depending on factors like how the employment agreement is written, "on the company's behalf" can be interpreted in different ways. The most common variables playing into this interpretation are time and equipment. Is the employee creating the work during working hours? Are they creating it during working hours and on company equipment? Are they creating it at any hour on company equipment? All three of these are possible interpretations of "on the company's behalf" and each one encompasses a different amount of creative works.

Use the [grid on page 31](#), marking in each box who owns the copyright for work created under each condition. Does the employee own it, or does your company? It's important to know the answer to this question.

Contributions, but on Whose Behalf?

It's important that all staff members of the company are aware of where the lines are drawn between works created on the company's behalf and works created by the employee on their own behalf. A common misconception here comes on the part of employees who don't realize that the code and contributions they make to FOSS projects may in fact be on the company's behalf. If this is so, then the company owns the copyright to those contributions;

		Equipment	
		Employee's	Company's
Time	Employee's		
	Company's		

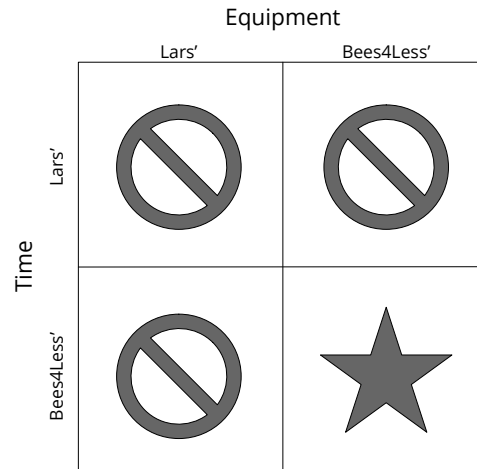
without explicit permission the employee is not authorized to distribute that copyrighted material. By doing so, they accidentally violate the copyright of the company and potentially initiate a whole bunch of legal headaches for everyone involved. Aside from the headache of violating the copyright of the company, by contributing to a FOSS project on behalf of but without permission from the company, a well-meaning employee may also publicly leak patentable intellectual property, company trade secrets, or strategic- and security-related information. As well, that contribution may be committing the company to a course of action without its knowledge or approval. These are all considerable business risks.

For example, Lars is the programmer who created the temperature monitoring software for Bees4Less. In the process of doing that work, he ended up adding new functionality to the pandas Python data analysis library⁹ and then contributing that functionality to the FOSS project, as shown in the [grid on page 32](#). Who owns the copyright to that new functionality?

According to the Bees4Less employment agreement, any intellectual property created on work time and/or on work equipment belongs to the company. Therefore, Bees4Less owns the copyright to the new functionality that Lars contributed to the FOSS project. Now the question is this: did Lars know he was supposed to get permission to contribute that code?

Don't. Panic. While it's not only possible but also likely that employees are contributing works created on behalf of the company to FOSS projects and that this is something that will need company attention, the odds are against

9. <https://pandas.pydata.org>



those contributions being especially risky. It's more likely that the contributions are innocuous, either by being so small as not to risk exposing any sensitive information or IP or by being made to a project that has no direct bearing upon the company's business model or operations.

For example, it's hardly a business risk to Bees4Less if, accidentally on behalf of the company, an employee makes a contribution to a FOSS project for transcribing guitar tablature. Instead, this is most likely an example of a company with an employment agreement written in such a way that "on behalf of" is interpreted overly broadly so that it encompasses works employees create on their own behalf.

Remember as well that your company likely relies upon a great deal of free and open source software to build and operate its business. If an employee must contribute a fix to a FOSS project to continue with the tasks they've been assigned by their manager, such as Lars did above, they might do so without considering the potential IP impacts of that contribution. All they want to do is their job, and removing a block by fixing a FOSS project allows them to do that.

So don't ignore unsanctioned contribution of company IP to FOSS projects, but don't overreact to them either. Typically they result from ignorance, not malice, and that ignorance isn't always on the part of the employee. Management ignorance of the sweeping scope of whatever part of the employment process defines that "on behalf of" phrase is often to blame. Those employment agreement/contract documents usually are old and haven't evolved along with either the times or the company's evolution. If you've not read the employment agreements being sent to those whom you're hiring, you should

take a moment to do so. You may be surprised by some of the clauses to which your employees are held.

Reviewing employment contracts and agreements for IP considerations is just one step in handling employee and company contributions to FOSS projects. You can take several more steps, thus converting a potential business risk into a strategic strength for the company. Achieving this requires education, intention, and attention. Thankfully, you're reading the right book for the job. You'll learn more about how to do this in [Chapter 17, Master Contribution Fundamentals](#), on page 263.

A Quick Word About Public Domain

While copyright “protects the owner of the exclusive property rights” for a work, those rights don't remain exclusive forever. All copyright, from the moment a work exists, has an expiration date (though what that date is varies by jurisdiction). Once the copyright expires, the work enters what's known as the *public domain*.

The official definition of public domain (PD) is murky at best, with inconsistent handling in the legal texts. Basically, works in the public domain no longer (or never did) have a copyright holder and are now free and open for anyone to use in whatever way they wish. While copyright expiration is the most common way for a work to end up in the public domain, some government and publicly produced or financed works start their lives that way. In other cases, a copyright holder can voluntarily relinquish rights to a work. The Creative Commons Zero (CC0) license allows a creator to place their work in the public domain in this way by declaring that they have “no rights reserved” (hence the Zero in the name).

Jurisdiction is a complication. That's because copyright can expire at different rates in different jurisdictions, sometimes even for different types of works (music rather than literature, for instance). This can lead to copyright expiring on a work in, for instance, Korea but not in the United States. Is that now a PD work? If you live in Korea, yes, but not so much in the United States. In some states, works that aren't produced there receive less copyright protection and may be considered public domain. For others, a foreign work's public domain status may depend upon whether the work was created in a country that's signed appropriate treaties. Some jurisdictions recognize that a creator can relinquish their rights, while others don't. Most of them are unclear about what to do should that creator change their mind and want their rights back.

The long and short of it is that when you start really looking at the matter of public domain, it's a hot mess. Thankfully, it's not something most companies need to worry about too much. If a work was created prior to 1850, licensed with CC0, or is an official act by the government, it's probably safe to use. And, as always, when in doubt check with your friendly neighborhood IP counsel.

Contributor License Agreements

An introduction to intellectual property concepts as they relate to FOSS wouldn't be complete without introducing Contributor License Agreements (CLA) and the Developer Certificate of Origin_ (DCO). You'll learn more about these in [Chapter 17, Master Contribution Fundamentals, on page 263](#), and [Chapter 20, Releasing FOSS Projects, on page 317](#), so we'll only lay the groundwork here to give you a solid footing later on.

With a name like “contributor license agreement” you will, I'm sure, be shocked to learn that a CLA is a legal document. Signed by a contributor prior to submitting work to a FOSS project, a CLA is an IP safety net for the FOSS project. Remember, FOSS projects are like coral: the combined copyrighted work of multiple individuals coming together to create a beautiful whole. The primary goal of a contributor license agreement is to minimize any potential legal complications from distributing those copyrighted works. By signing a CLA, the project contributor affirms the following:

1. They have the right to share the IP that's in the contribution.
2. They give the project license to alter, distribute, and administer their contribution.
3. The contributor will never revoke that license.

Occasionally a CLA will also include a transfer of the copyright from the contributor to the project or the project's governing body. This copyright reassignment via CLA is relatively rare but happens often enough that you should keep an eye out for it.

Knowing what you now do about the jagged rocks hidden in the sea of intellectual property, you would think that having a safety net like a CLA would be a no-brainer, yet they're not especially common in the FOSS ecosystem. We're starting to see them with increasing frequency in large FOSS projects, especially those under the aegis of large nonprofit foundations or large corporations.

Many FOSS projects and contributors object to CLAs (sometimes vehemently). To them, the requirement to sign a CLA prior to contributing is a barrier to entry that unnecessarily slows down and complicates the contribution process, thereby discouraging contributions. Some FOSS contributors distrust CLAs, viewing them as a way for corporations to take advantage of the free work of the community or even to steal the work by changing to use a proprietary

license. These (often real) concerns can cause a lot of friction in projects that require signing a CLA prior to accepting a contribution.

For businesses, CLAs can cause problems as well-meaning employees sign and contribute without understanding that they're doing so on behalf of the company. It's unlikely that the contributing employee is authorized to be signing any sort of legal documents involving the intellectual property of the company. This behavior can expose the company to risk or, at the very least, to an irritating project for the legal team to clean up.

Don't forget the administrative overhead of collecting and maintaining the signed documents along with verifying that there's a signed CLA for every single contribution going forward. This CLA maintenance has a security and privacy aspect to consider as well since these documents include personally identifying information (PII) that must be handled appropriately.

Overall, CLAs can serve an important purpose for some projects, but they also can be a big pile of headaches. An IP safety net would be handy, but surely there's a better way?

Developer Certificate of Origin

Enter the Developer Certificate of Origin (DCO),¹⁰ created from the ground up to address some of these concerns about CLAs. The DCO is also a legal document but is short and easy to understand:

Developer Certificate of Origin Version 1.1

Copyright (C) 2004, 2006 The Linux Foundation and its contributors.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or

(b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or

10. <https://developercertificate.org/>

(c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.

(d) I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

To agree to the DCO, a software developer must sign their contribution using the `-s` or `--signoff` flag on their commit in the Git version control system or use a similar form of attestation. By signing in this way, the contributor confirms that they have the right to distribute their contribution and that they do so under the same conditions as the license of the project. Their signature—which must be on every contribution but is simple for a developer to include automatically—then becomes a part of public record in the version control repository for the project. The simple process addresses many of the concerns projects and contributors have with CLAs, but it's not perfect. By requiring the signature be made in this way, in practice the process ends up limited to those contributions made using Git. Projects that use other version control systems, or contributions that are not made via Git, can use the DCO, but it's a more complicated process. These projects may need to fall back to using a CLA if they feel the need for the IP safety net.

Whether a project uses a CLA or DCO is something your company should be keeping an eye on. Often, if one of these safety nets is in place, well-intentioned employees eager to make a contribution and get on with their work will sign without understanding the impact it may have on a company. By signing either of these, the employee is saying that they have permission to share that copyrighted work—regardless of whether they've actually *received* that permission. Some CLA-using projects have instituted two different forms of CLAs to help clarify on whose behalf someone is contributing the work, their own (Individual CLA, ICLA) or their employer (Corporate CLA, CCLA). The CCLA has the advantage of being signed once by a company and covering contributions from all of its employees, reducing some of the overhead. The Apache Software Foundation is one of the organizations that have both an ICLA and a CCLA. The Apache CCLA makes it clear that the company signing the agreement is in control (“designated employees”) of who is authorized to make contributions on its behalf:

This version of the Agreement allows an entity (the “Corporation”) to submit Contributions to the Foundation, to authorize Contributions submitted by its designated employees to the Foundation, and to grant copyright and patent licenses thereto.

Regardless of whether a project uses a CLA or a DCO, it can be very difficult for you to see whether an employee has signed either document to contribute work on behalf of the company. It's important to make sure not only that employees know the right thing to do when faced with a CLA or DCO but also why it matters to them and to the company at large. Having a policy in place, well communicated and easy to find, can go a long way to minimizing potential IP mistakes. You'll learn about this in [Chapter 9, Establish Effective Inbound Policies and Procedures, on page 145](#).

Now that we've set the intellectual property stage, it's time to introduce the star of the FOSS show: licensing.

Licenses: The Rules of IP Engagement

As you learned in [Chapter 2, Make the Intangible Profitable with Intellectual Property, on page 17](#), for the most part copyright is something that Just Happens whenever an idea is expressed in a fixed form. It's automatic the moment a work is fixed, with no further action generally necessary. Giving permission for any sort of use of the work, however, must be intentional and deliberate. A copyright holder must take explicit action to confirm that, yes, these types of actions are permitted on the work under those types of conditions. That action is known as a license.

This chapter introduces you to the concept of licensing and the different types of software and FOSS licenses that exist in the ecosystem, but it won't touch upon license compliance yet. That's a complex and important topic. License violations, whether for proprietary software or for FOSS, can end up costing millions of dollars on top of the additional costs of brand damage, and potentially can even affect how and whether your company releases products or does business (as you'll learn later, much of this is scaremongering). While license compliance can't be ignored, a bit of intention and attention can go a long way, and most violations can be remediated amicably (especially if caught early). You'll learn all about license compliance and how to manage it later on in [Chapter 12, Maintain FOSS License Compliance, on page 201](#).

Basics of Licensing

What exactly is a license? In the context of intellectual property, the license declares what can and can't be done with a copyrighted work, by whom, and under what conditions. Because they're giving permissions for intellectual property, licenses are legal instruments that are enforceable by law. While they don't necessarily need to be formal or even be written down at all, it's difficult and risky to try to enforce an oral license.

The instant a work exists in fixed form, the creator (also known as an “author” in legalese, even if the work isn’t literary) has all the rights to it. Only the copyright owner can use, copy, modify, distribute, or otherwise take action on or with that work. This is called All Rights Reserved. You’ve probably seen it printed on things but never really considered what it means. Because copyright is immediate, it’s not necessary for those authors to specify all rights are reserved on the work since it’s the default case, but in matters like this it never hurts to be explicit about permissions (or, in this case, lack thereof).

If there are explicit permissions, they’re in a license. Licenses can include any sort of permissions or conditions the author sees fit to include. They also can be limited to individual people or organizations (exclusive license), categories of people or organizations (only nonprofits, for instance), or open to anyone and everyone. FOSS licenses fall into this last category, as they cannot limit who may make use of the work. Because licenses can include any sort of permissions or conditions, it’s possible to create a license that’s frankly silly. An author could limit use of their work to only those people born on a Tuesday under a full moon, and it could be a valid license. That validity is enhanced by someone *agreeing* to the license. If I want to use your work and I agree that I will only do so when at least five meters below sea level, then however silly it may be, I am bound by that agreement if I use your work. If you learn that I used it when I was atop Machu Picchu, then you as the author are within your rights to sue me for violating your copyright. Remember, licenses are *legal documents* even if they are silly and not drafted by a lawyer. It can be easy to forget this in the FOSS world, which sometimes looks like an “anything goes” environment.

Silly or not, all FOSS licenses vary in length and complexity, depending upon the permissions and obligations the license drafter wished to convey. Some of them, such as the GNU General Public License version 3 (GPLv3) or the Apache Software License version 2 (Apache-2.0) extend for pages of understandable yet dense legal text. Others, like the 3-Clause BSD License (BSD-3-Clause) shown here, get their point across in just a couple of paragraphs:

Copyright (c) <year> <owner>.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This license states that people can do whatever they want with the software and its source code as long as they meet the three criteria listed there. It also includes—in somewhat shouty capital letters—a *warranty clause* or, more specifically, a *no-warranty clause* declaring that the copyright holders are in no way responsible for anything that should happen should anyone use the software or its source code for any purpose whatsoever. This type of clause is a standard feature of most FOSS licenses regardless of the permissions and conditions that the license contains.

The Contractual Controversy

So if licenses are legally binding, are they contracts? Well, it depends on whom you ask. In the case of FOSS licenses, the question is contentious. Some legal experts believe that these are *bare licenses*, which is to say they're solely nonexclusive copyright licenses that provide blanket, declared permissions to anyone. And this is true! Free and open source licenses definitely contain nonexclusive permissions to copyrighted intellectual property. This is part of what makes them work so well.

Other legal experts believe that a license is a contract, where the thing that the contract covers is the copyrights, defining various obligations of the licensee to the licensor. After all, if a court is to judge upon license enforcement, it will need to employ several contractual concepts to do so. Still others believe that a FOSS license is a bare license (purely copyright permissions) with contractual components mixed in. The answer to this question can matter a great deal in copyright infringement cases. For instance, in the United States a breach of contract is a matter for the state courts to decide, but copyright license infringement is a federal matter, so whether the lawyers

and judge interpret violated license clauses as bearing on contract law or copyright law can have an oversized impact on proceedings.

Regardless of interpretation, a license still represents permissions and an agreement, and violating it can land your company in a lot of legal hot water. It's always best to treat them with the same level of respect you would any other legal agreement—and always consult legal counsel when questions arise.

What's up with Fair Use?

It doesn't come up too often in FOSS, but it wouldn't be fair to ignore fair use when discussing IP-related matters.

In almost all cases, the only way to use a copyrighted work is to have permission (a license) to do so. International copyright law recognizes that this can make it difficult for some people to operate, including educators, journalists, researchers, comedians, and other who engage in forms of education, commentary, or criticism. Fair use (also known in some jurisdictions as “fair dealing”) acknowledges that sometimes it's OK for members of the public to fairly use portions of a copyrighted work for these purposes even if they don't have permission from the copyright owner.

The definition of “fair use” varies by jurisdiction. Even within a single jurisdiction, the guidelines often are intentionally broad to allow maximum flexibility. A judge then determines on a case-by-case basis whether a use is “fair” or not. Despite differences by jurisdiction, this determination tends to follow these guidelines:

- What did you do with the copyrighted work? Did you use it as is or did you change it in some way to add new or different meaning (create a *transformative work*)? Transformative works are more likely to be judged as fair use.
- How are you going to use the copyrighted work? Will it be for commercial purposes? If so, it's less likely to be judged as fair use.
- What type of copyrighted work are you going to use? Is it factual (nonfiction) or something more creative (fiction)? Use of factual works is more likely to be judged as fair use.
- How much of the copyrighted work are you using, and is it the “heart” of the work? If you're using a lot of the work or a small bit but it's “the good stuff,” then it's less likely to be judged as fair use.
- Is your use going to impact the future market or value of the copyrighted work? If a judge thinks that it will, then it's unlikely to be fair use.

These are only the most common guidelines a judge might use, and every case is different, but if you need to use a portion of a copyrighted work without permission, then you might want to consult with your IP counsel to see whether fair use could apply in your situation.

Licensing Innovation: FOSS's Killer App

Despite just anyone being able to “write” a license for their works, please don't. This is one of those situations where just because someone can do something doesn't mean they should. A lot of specialized language goes into properly drafted licenses, and should matters go to court, the judge will have certain expectations that an amateur license-drafter won't have considered. Specialist license drafters, also known as IP lawyers, are required. Yes, they are back.

Prior to the invention of free and open source software licenses, life was a lot more complicated for software developers. Any time someone wanted to release a piece of software, they'd either need to have their lawyers draft a new licensing agreement for it or they could put the software out there and assume the right thing would magically happen (the *YOLO licensing strategy*). Each license was a lovingly handcrafted, bespoke, artisanal piece of legal work. However fancy or effective the license, there's no denying that creating it required a great deal of expertise, time, and especially money. Spending a lot of money and waiting for the license drafting to complete before you can ship your software? Surely there's a better way?

FOSS licenses are that better way and are the “killer app” of free and open source software. Rather than waiting for a lawyer to draft a custom license for the software, you can select from a collection of professionally drafted and community-accepted FOSS licenses. Instead of it taking weeks to receive a license and release the software, with FOSS licenses it can now be done if not instantaneously, then at least in a matter of minutes or hours. In many cases, this can all occur without having to involve a single lawyer.

This is possible because these licenses are standardized and relatively well understood. In the FOSS world, you can say to someone, “this project is licensed BSD-3-Clause” and most everyone will have some idea what that means with regard to the permissions and conditions related to using that FOSS project. Sometimes it might be necessary to have a lawyer do a close reading of the license or to consult the license steward for clarification of the intention of certain license clauses, but thanks to the standardization, these situations are special cases rather than a matter of day-to-day operation, as they are with non-FOSS (proprietary) licenses. In this way, most individuals often can get by with using free and open source software projects without ever needing to consult a lawyer.

Of course, everything is a bit more complicated in a corporate setting. The need for lawyers is more pressing in this situation. It's foolish and probably counts as business malpractice to operate a company without the benefit of legal counsel, especially where intellectual property matters are concerned. We'll talk about this in detail in [Chapter 12, Maintain FOSS License Compliance, on page 201](#), and [Chapter 20, Releasing FOSS Projects, on page 317](#), but whether using or releasing software, license selection can have massive long-term impact on business strategy. While it may seem expensive now, collaborating with an IP lawyer to review or perhaps create your company's FOSS license policies can have a large payoff down the road and is a good investment. These few hours of work can save the company from accidental license infringement, releasing critical intellectual property, or a copy and relaunch (a *fork*) of a FOSS project the company released, see also the [sidebar on page 45](#).

Types of FOSS Licenses

Well over a hundred FOSS licenses are out there in the world today, and generally they fall into one of two different buckets: reciprocal or permissive.

Reciprocal (aka Copyleft) Licenses

Historically, the first ever FOSS licenses were the reciprocal ones. Also often known as *copyleft* licenses (a play on the word *copyright*), reciprocal licenses are most frequently associated with the Free Software side of free and open source software because they place the Four Freedoms of Free Software—the freedoms to run, copy, study, and modify—front and center. In this book I'll be using the terms *reciprocal* and *copyleft* interchangeably, with a preference for *reciprocal*, as it's descriptive rather than merely clever.

Despite there being fewer reciprocal licenses than the permissive ones you'll learn about next, these licenses are popular. The Mozilla Public License is one example, but the reciprocal licenses you'll see most often belong to the GNU General Public License (GPL) family: GPL versions 2 and 3 (GPLv2, GPLv3), Lesser GNU General Public License (LGPL), and Affero General Public License (AGPL). The Free Software Foundation (FSF) is the steward for all GPL licenses, which are used by powerhouse projects such as the Linux kernel and the Blender 3D graphics applications.

From the outset, reciprocal licenses took the established practices of copyright—seen by some to restrict innovation and user freedom—and turned them on their head, using them instead to *ensure* innovation and freedom. This is the source of the word *copyleft*: it's using copyright to free works rather

Do You Have the Spoons to Deal with a Fork?

To “fork” a FOSS project means to take a copy of the project, rename it, and start a new project and community around that copy. Those who fork a project rarely, if ever, contribute to the parent project again. It’s the software equivalent of the Robert Frost poem: *Two paths diverged in a codebase and I, I took the one less traveled by...and that has made all the difference.*

There can be many reasons for a project fork. Perhaps the project has lain dormant for a while and someone wants to revive it. Perhaps the company that has underwritten the project has been acquired and the community is afraid that the new parent company may close the project. Or perhaps a schism within the community itself prompted a portion of the community to go a different direction with the project. Often a project fork is accompanied by a great deal of discussion and possibly also community strife. Whatever the reason, a project fork is the copying of a project with the purpose of creating a new and separate community around it. While the fork does require some technical work, it’s primarily a social action.

There have been many forks throughout the history of free and open source software. Some notable ones are OpenTofu forking from HashiCorp’s Terraform, Jenkins forking from Hudson, and Amazon forking OpenSearch from Elastic’s Elasticsearch.

Nothing is ever simple with free and open source software, so naturally there’s a complication to this entire process. When GitHub launched back in 2008, it chose the word fork to represent the action of a Git clone command. When you fork a project on GitHub, you’re actually creating a clone of it—a copy on which you can perform your work. It’s entirely possible that from here you may choose to fork the project in the original sense: create a separate project and associated community rather than simply sending pull requests back to the original project. However, nearly all people who fork a GitHub project only intend to create a personal working copy, a clone.

This overloading of the word fork has caused more than a little confusion in free and open source software communities. As you’re working with either inbound or outbound FOSS, if you come across a project that someone calls a fork, trust but verify. Have a close look at the project to make sure it isn’t only a clone.

The title of this sidebar is a reference to Spoon Theory, a handy metaphor popularized by Christine Miserandino in 2003.^a

a. https://en.wikipedia.org/wiki/Spoon_theory

than lock them up. Software that’s based upon or incorporates FOSS projects that are under reciprocal licenses *must* be released under those same permissions and conditions. This is where the reciprocity comes in: as you have benefitted from this FOSS project, so must others similarly benefit from your own work if that work incorporates the project. The permissions and conditions of these licenses are like the software IP equivalent of the Golden Rule: do

unto others as you would have them do unto you—or as you have done, thanks to them. You have received and used the reciprocal software under certain terms and must ensure that others can use all derivative software under the same terms (more on what constitutes *derivatives* shortly).

For those who aren't paying attention to license terms, those of reciprocal licenses can come as an unpleasant shock. “What do you *mean* that, having created my software product around this reciprocally licensed FOSS project, I'm now *required* to offer the source code for my product to everyone for free and under the same license? Seriously??” This is the sort of surprise that can derail business strategies and has earned reciprocal licenses the unfortunate nickname of “viral licenses,” because once you catch one, you pass it on to everyone who interacts with your product. It's a pejorative term that has caused a great deal of damage to the brand of reciprocal licenses. In truth, as you'll see in [Chapter 20, Releasing FOSS Projects, on page 317](#), reciprocal licenses can offer a lot of value to your product and business strategies.

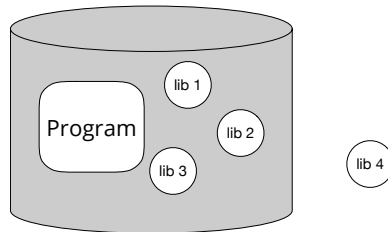
In 2019, reciprocal licensing received its first innovation in more than ten years with the introduction of the Cryptographic Autonomy License (CAL-1.0), which not only requires the sharing of source code modifications (reciprocity) but also requires that people have access to their own data as well as any cryptographic keys required to encode or decode it. CAL-1.0 is the first time an open source license has ensured a user's rights to their own data. It's too early in the license's life cycle to be able to gauge the impact that this right may have on the software ecosystem.

Strong and Weak Copyleft

Copyleft was invented to encourage and ensure the spread of Free Software and the Four Freedoms for all software developers and users. GPLv3 is an example of a license designed to maximize that spread. Any software that uses even a piece of a project that's under the GPLv3 license must itself be released under GPLv3 and also provide its source code to its users. Early on in the history of reciprocal copyleft licensing, it became obvious that this enforced reciprocity was going to make it difficult to combine software under these licenses with those under others (open source or not), and this would in some cases force programmers to choose proprietary solutions rather than Free Software ones. Free Software would be working against its own goals.

The problem came to light thanks to programming concepts known as *libraries* and *linking*. Libraries are aptly named, as they're external collections of functionality that programmers can use to make it quicker and easier to develop software. A program can use the functionality of a software library

by linking to it. This linking can either be *static*, where in essence the entire library is slurped up into the program and becomes a part of it during software compilation, or *dynamic*, where the program is compiled with signposts pointing it at the location of the library so it can grab the necessary functionality as needed when it runs. The library must be available to the program in some way, either shipped with the library alongside or already installed and findable by the program (otherwise it can't locate the functionality when it needs it), but the library doesn't become a part of the program itself as it does in the case of static linking.



For instance, in the preceding image the libraries one through three are easily available to the program. Either the program shipped with the necessary libraries or the program can “see” the library and knows where to find it when it’s needed. Library four, on the other hand, may be installed but is outside of the area that the program knows about. It’s invisible to the program and therefore unavailable when its functionality is needed.

Some programming languages, such as the popular and venerable C and C++, have standard libraries that provide an enormous amount of functionality that’s used for nearly every C or C++ program, such as math utilities, printing, and file access. These standard libraries are critical for much of the software that exists today. The GNU Project developed its own Free Software version of the C standard library. It provided the same functionality of expensive proprietary standard libraries, but it was free in all senses of the word. This naturally made it very appealing to software developers, but many were finding they were unable to use it. Under the terms of the GPL, even dynamically linking to a library was considered a *derivative* of the GPL’d code, so the resulting program would have to be released as GPL even though no GPL’d code was officially inside the compiled program (as it would be were the library linked statically). Due to the business requirements of the software that they were building, these programmers couldn’t allow their programs and code to be released under the GPL license. In this case, the reciprocity requirements of the license were preventing the spread of Free Software across the C and C++ ecosystems. You’ll learn more about derivatives in [Derivatives, on page 58](#), later in this chapter.

Enter the Lesser GNU General Public License (LGPL), the first of many licenses known as *weak copyleft*. These licenses are released with reciprocal license terms but have license qualifiers allowing them to be *used* without triggering the reciprocal terms of the license. Changes to a piece of software under a weak copyleft license must still be released under the same license terms, but developers and others may use the functionality of the software in their own code, which they can then release under any license terms that they wish. The relevant part of LGPL-3.0 for this is section 3:

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

If instead of using the LGPL'd software functionality a programmer *combines* their program with the LGPL's software, then it's a new work that's a derivative of the LGPL'd work and is subject to the terms of reciprocity in the license. Section 4 of LGPL-v3.0 covers this situation and includes reciprocity terms, such as providing attribution, sharing source code, and providing installation instructions, among others.

LGPL was the first “weak” copyleft license, but since its invention several others have popped up, including the Mozilla Public License (MPL) and the Eclipse Public License (EPL). The license terms of each weak copyleft license are different, but each one is reciprocal in nature and includes some way to use the licensed work such that the resulting work is not subject to the reciprocity terms.

Network Reciprocal/Copyleft

The final type of reciprocal license you're likely to see (or use!) is the *network reciprocal* license, more commonly known as the GNU Affero General Public License or AGPL.

The AGPL is basically the same as the GPL but with a special case for distribution. You'll learn more about the common forms of distribution later in the chapter in [Distribution, on page 60](#), and those certainly do apply to the AGPL

as well. However, the AGPL also considers making the software available over a network—such as in a software-as-a-service (SaaS) situation—to be distribution:

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software.

Due to this clause, the AGPL has become a popular license for open core software startups. You'll learn more about open core in [Chapter 22, Incorporate FOSS into Business Models, on page 369](#).

Permissive Licenses

The other bucket for FOSS licenses is Permissive. Like *reciprocal*, the term *permissive* here is descriptive: by largely removing the requirements for reciprocity, these licenses permit people to do different things with a project than a reciprocal license would. Whereas works that incorporate reciprocally licensed FOSS projects must be released under licenses that retain that reciprocity forever, works incorporating FOSS projects released under permissive licenses can do whatever they like with their final work as long as they meet the license conditions. They may release the work's source code under the same license, under a different license, or even not release it at all. If the work is distributed, it typically must include the original copyright and license information but may itself be released under a different license.

Having fewer constraints has allowed the number of permissive licenses to flourish, such that they now make up the bulk of all FOSS licenses. Being populous naturally means that these licenses are also popular and frequently dominate research reports about the commonality of various licenses in the FOSS ecosystem. There is, after all, strength in numbers. The MIT License is especially popular lately, perhaps due to its brevity, clarity, and simplicity. Weighing in at a mere four sentences, this license allows entirely unfettered use and redistribution of the software with the sole condition that the copyright and license statement be retained. Similarly popular but at the other end of the brevity spectrum, in [Patents, on page 21](#), you learned about the Apache License Version 2.0 (Apache-2.0)'s explicit patent clause, which has made it a favorite of businesses:

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

Translated, this legalese essentially says that if any patents apply to a contribution either on its own or in combination with the rest of the software, the contributor who submitted it gives everyone license to use those patents. This means that others can use the software without fear of being sued for patent infringement, and companies think this is pretty great.

You might in fact be so charmed by this patent clause that your company is tempted to set a policy that any FOSS project they release *must* be under Apache-2.0. This may sound like a good idea, but as you'll learn in [Chapter 16, Strategic Benefits and Risks of Outbound FOSS, on page 243](#), blanket policies like this can obscure strategic business moves. For some companies, rather than a single *prix fixe* license it may make more sense to offer an *à la carte* limited selection and clear guidelines for when each one should be used. Whether this is the case for your company is something you'll learn more about later in the book.

What Makes a FOSS License “Open Source”

A couple times now I've mentioned that there are over a hundred open source licenses. This, of course, implies that there are *criteria* for being an open source license and also someone who judges those criteria. Both of these implications are true. The criteria are the aforementioned Open Source Definition (OSD) and the someones who certify whether a license meets those criteria are the Open Source Initiative (OSI).

When an individual or organization has a license that they believe qualifies to be certified as open source, they can submit it to OSI's license review process.¹ Through public discussion, a community of legal and FOSS experts as

1. <https://opensource.org/licenses/review-process/>

well as OSI members and directors of the board examine the license in detail, comparing it against the OSD and verifying that it’s well written. To qualify as an open source license, it must obey each and every one of the ten items in the Open Source Definition, which you learned about in [The Dawn of Open Source Software, on page 9](#). Violating even one of these is grounds for rejection:

1. The project must be freely redistributable—even if sold and even if it’s part of a larger collection of software.
2. All source code must be available and distributable.
3. Modifications and derived works must be allowed and distributable under the same license terms.
4. If distribution of modified code isn’t allowed, it cannot prevent distribution of patch files (snippets of source code that can be applied to include new fixes or functionality) along with the unmodified code.
5. In no way can the license under which the code is distributed discriminate against any person or group. All people must be allowed to use the code on the same terms, even if they’re bad people.
6. Similarly, the license also can’t single out industries, companies, or other types of undertakings. All groups and ventures must be allowed to use the code on the same terms, even if those groups condone horrible things.
7. The license applies to anyone who receives a copy of the software without needing any additional permissions.
8. The license can’t restrict someone from extracting the project or code from a larger collection. If they do extract it, it’s available to them under the same license terms as the whole, larger collection.
9. If the software is distributed as part of a larger collection of code, the license can’t put any restrictions or requirements on that other code.
10. The license applies to all technology and user interface applications of the software to which it’s applied.

Licenses that have not been drafted or at least reviewed and edited by legal professionals can also be rejected, since these documents often contain nonstandard language and legally confusing (and therefore legally risky) ambiguities. Some people who participate in the review process will refer to these by the derogatory term *crayon licenses*, implying that the author or authors lack the experience or knowledge to draft license documents.

Once the license review committee has completed its evaluation, it makes its recommendation to OSI: accept the license and add it to the list of certified open source licenses or reject the license for the stated reasons. The OSI Board of Directors reviews these recommendations at their board meetings, almost invariably accepting the recommendations of the committee. Because of the amount of discussion required, as well as the possible need for document revisions by the license steward, it's common for the review process to extend over multiple months.

Upon approval, licenses are added to the list maintained at <https://opensource.org/licenses>. Licenses that do not appear on this list cannot be guaranteed to follow the standard of the OSD and should not be called “open source.”

Why This Matters

Right about now you may be thinking, “Yes, yes. It's bureaucracy and policies and procedures for licenses. I don't see why I should care about this,” which is fair enough. At first glance, the license review process and OSI-approved license list may not appear to impact your company or business, but look at it little deeper and you'll see just how important this process is.

As you learned in [The Importance of Standards, on page 14](#), it all goes back to the importance of standards. For any important standard in our world, you'll find a body of people tasked with certifying that items follow that standard. How far do you think you'd get with interoperability if, for instance, every manufacturer of devices decided to use their own version of USB? If you unexpectedly learn that the USB cable that came with your camera won't work at all with your phone, you're likely to be justifiably irritated. Which is doing it wrong, the camera, the phone, or both? Without adherence to standards, there's no way for most people to tell.

The same holds true for open source licenses. Having a single body—the Open Source Initiative—dedicated to reviewing and certifying licenses not only ensures a moderately consistent interpretation of the standard but also removes the necessity of their effort being duplicated by countless corporate lawyers. So important is this certification and list of approved licenses that the phrase “OSI-approved licenses” is enshrined in many corporate IP and open source policies as well as merger and acquisition processes. If the software supply chain of the company includes licenses that are not OSI-approved, policies may have been broken somewhere along the line, and additional scrutiny is required.

Summary of Some Common FOSS Licenses

Don't worry if you're feeling a bit overwhelmed at this point. Your IP lawyer friends read stuff like this all day long, so you now have an idea of how hard their jobs can be.

The situation isn't as bad as it seems though. While a lot of FOSS licenses are out there, for the most part you'll only ever deal with a handful of them. Here are some of the most common free and open source licenses in the ecosystem, along with a brief summary of their attributes.

Name	Type	Summary
GNU Affero General Public License (AGPL-3.0)	Strong network reciprocal	Preserve copyright/license statements, distribute source code under same license, network availability equals distribution, explicit patent clause
GNU General Public License (GPL-3.0)	Strong reciprocal	Preserve copyright/license statements, distribute source code under same license, explicit patent clause
GNU Lesser General Public License (LGPL-3.0)	Weak reciprocal	Preserve copyright/license statements, distribute source code for modifications under same license, dynamically linked libraries excluded from reciprocity
Mozilla Public License (MPL-2.0)	Weak reciprocal	Preserve copyright/license statements, distribute source code for modifications under same license, may distribute MPL components with proprietary if kept separate, explicit patent clause
Eclipse Public License (EPL-2.0)	Weak reciprocal	Preserve copyright/license statements, distribute source code for modifications under same license, may distribute EPL components with proprietary if kept separate, explicit patent clause
Apache License (Apache-2.0)	Permissive	Preserve copyright/license statements, declare any changes to original code, include NOTICES attributions, no requirement to distribute source code, relicense OK, explicit patent clause
MIT License (MIT)	Permissive	Preserve copyright/license statements, relicense OK
BSD 3-Clause "New" or "Revised" License (BSD-3-Clause)	Permissive	Preserve copyright/license statements, relicense OK, can't use trademarks

Notice that while each license is different (why else have so many?), they all share many of the same characteristics. For instance, most FOSS licenses require that any modified and distributed software retain and include any existing copyright and license statements that are already in the code. This makes sense and is the polite thing to do, anyway. After all, if you made something and shared it with others, you'd like to receive credit for it too.

Other Types of Licenses

When someone mentions licensing, it's the open source software licenses that grab the limelight these days. It's as though people have forgotten (or perhaps never knew) that permission is required to use any sort of intellectual property, and therefore licenses are everywhere and can apply to all sorts of things. Depending upon your business needs, additional types of non-software open licenses may be useful.

Creative Works

The concept of free and open access extends beyond software. The Creative Commons (CC) suite of licenses, for instance, are designed to cover many different types of creative works. Visual art, textual content, audio works, and even data benefit from these popular open content licenses. Stewarded by the Creative Commons nonprofit organization,² CC licenses leverage standard and composable permissions and conditions to create a variety of licensing options:

- Attribution (BY): When distributing the work, you must credit creators who contributed to it.
- Share Alike (SA): Any work must be distributed under the same terms and conditions as those you received with the original work.
- No Derivatives (ND): While you may use the work, you may not modify or otherwise create derivatives from it.
- No Commercial (NC): You may use the work for any purpose *except* commercial ones. Works under an NC license can therefore not be used by your company for anything related to its for-profit business operations.

Creative Commons licenses are referred to by their abbreviations: CC BY, CC BY-SA, CC BY-NC, and so forth. The most commonly used CC licenses are CC BY (Creative Commons Attribution, a very permissive license à la MIT for software) and CC BY-SA (Creative Commons Attribution Share Alike, the CC

2. <https://creativecommons.org/>

version of a reciprocal license). The Creative Commons organization also provides a CC0 (Creative Commons Zero) license, which can be used for an effect very much like placing a work in the public domain. With the exception of CC0, Creative Commons licenses should never be used for software source code.

Data and Hardware

Though CC licenses can be used for releasing data, the Open Knowledge Foundation (OKFN) provides its own suite of licenses specifically designed for data. Like FOSS and Creative Commons licenses, these Open Data Commons (ODC) licenses also come in permissive, reciprocal, and public domain versions. The Open Source Hardware Association (OSHW) maintains the definition and best practices for openly designing and releasing hardware, including guidance for licensing of the intellectual property materials necessary to create the hardware. CERN, the European Organization for Nuclear Research, has created the CERN Open Hardware License (OHL) to help encourage the sharing of designs both inside and out of scientific research labs. As has become the norm, the CERN Open Hardware Licenses are available in both reciprocal and permissive variants, showing the enduring impact that the FOSS movement has had on other creative endeavors.

Proprietary

Of course “open” isn’t the only way to go with licensing. A company might have plenty of reasons to want to keep its IP under more strict licenses, and without the constraints of the Open Source Definition a company can explore different permission options. For instance, Unity Technologies, maker of the Unity 3D gaming engine software, provides some of the source code for their product under the Unity Reference-Only License.³ This allows developers to create custom tooling that’s tightly integrated with Unity’s software but not to change that software in any way. This has enabled Unity to build a diverse ecosystem of utilities that not only help speed game development but also increase the costs of switching to another platform.

Most of the software you use on a day-to-day basis is provided to you under closed-source—also known as proprietary—licenses. These End User License Agreements (EULAs) are those pages and pages of confusing terms and conditions to which you’re required to agree before you can install or even use many software products. If you’re like most people, you click Agree without reading the document, so it may surprise you to learn that in many cases,

3. <https://unity.com/legal/licenses/unity-reference-only-license>

when you “purchase” the software you are instead paying a licensing fee and don’t actually own the program at all. The company providing the software retains ownership over the software itself. In this way the company can prevent people from sharing or reselling the software, even if the consumer is no longer using it. Similarly, online services require consumers to agree to terms of service and other conditions. This also is a form of licensing, since it’s setting out the permissions under which someone is allowed to use the service.

Fauxpen Source

Finally, we’ll look at the “fauxpen” (faux open) source software licenses. A relatively new development in the software world, the stewards of these proprietary licenses frequently claim them to be “open” but carefully avoid describing them as “open source” as none of them have proved to adhere to the Open Source Definition or been approved by the Open Source Initiative. Thus far fauxpen source licenses primarily appear when a company formed around and controlling a software project that’s released under an OSI-approved license decides that this licensing strategy is no longer in their best business interests. In an attempt to retain the community and cache that come with an open source project, these companies create and apply new licenses that are *source available*—the source code is open for inspection—but restrict the use, modification, or distribution of the software in some way.

The most common fauxpen source licenses at this time are the Business Source License (BUSL-1.1) and the Server Side Public License (SSPL-1.0). Software released under BUSL cannot be used “in production,” a vague term that the licensor may choose to refine in a clause in the license. Four years after the software is released under BUSL, it will automatically revert to being available under a GPL-compatible open source license.

SSPL is similar to AGPL in that it’s a license that hinges upon whether software is distributed over a network as SaaS. If a licensor, like your company, does provide an SSPL-licensed software as an online service, then they’re required to release every piece of the software stack required to run the service, and all of this software must be released under the SSPL license. From a business perspective, this could be exactly as bad as it sounds. Also, some vague terms in the license, depending upon how they’re interpreted, could mean a license violation would impact an immense amount of software. For instance, according to the text of the license, it’s unclear where that software stack ends. If the software runs in a Windows virtual machine in the Microsoft Azure cloud, does that mean the licensor would be required to release the Windows operating system under the SSPL? How precisely are they supposed

to do that if—as is likely the case—they don’t have relicensing authority for Windows? None of these questions are necessarily the problem of the organization that released the software under SSPL. Instead, it’s the licensee who must take care when using the SSPL’d software.

Not necessarily problematic in themselves, fauxpen source licenses cause complications for companies that had included the open source version of the project in their software supply chains. Suddenly the license terms to which the company agreed have changed, and for the sake of the business the company must reevaluate their usage of the project as well as review the new license. Failure to do so, usually by not noticing that the project has been relicensed, can obligate the company to terms and conditions whether they know it or not and therefore increase risk to the business.

Too Much of a Good Thing?

The owner of copyright over a work may license it however and to whomever they choose. A painter can give a publisher a license to use their artwork on the cover of a book and nothing else, then give a clothing designer a license to use the same artwork on a pair of shoes. They can mix and match different licenses depending upon the occasion and opportunity, and they can charge a license fee each time if they wish.

The copyright owners of a FOSS project can also choose to release it under multiple licenses if they want. When this happens, it’s most often because the project is available under a reciprocal license like GPL or AGPL but someone wishes to use the project without the reciprocity obligations of these licenses. The copyright owners can then provide a nonreciprocal license—often for a subscription fee—to that user.

Some projects are released under more than one FOSS license for license compatibility reasons. For instance, Rust is a FOSS programming license that’s released under both the MIT and Apache-2.0 licenses.^a The Rust community did this so the language would be as compatible as possible with all other FOSS licenses and use cases. The MIT License is highly permissive and considered compatible with reciprocally licensed software. Apache-2.0 isn’t compatible with all reciprocally licensed FOSS, but it does have a patent clause that companies like. Releasing Rust in this way means that users gain the benefits of both licenses at once.

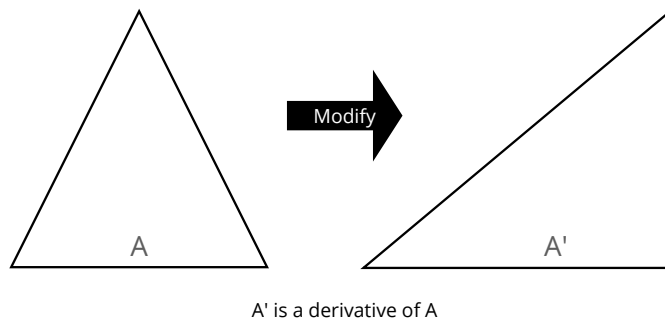
Keep an eye out for projects released under multiple licenses, and if you use any, bring them to the attention of your Legal and Open Source Program Office (OSPO) teams. They’ll be able to help sort out the license obligations. You’ll learn more about the OSPO in [Chapter 6, Introducing the Open Source Program Office, on page 101](#).

a. <https://github.com/rust-lang/rust/blob/master/COPYRIGHT>

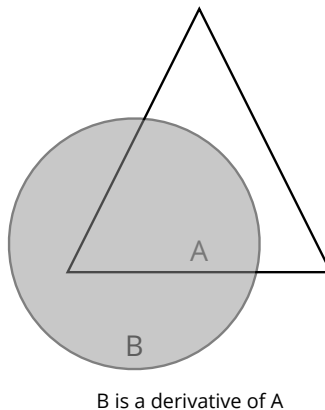
Derivatives

To some, a “derivative” is something you studied in calculus. To others, it’s a type of financial product. For people in FOSS, a derivative is a modification to an existing copyrighted work or a work that incorporates some or all of another copyrighted work. You’ve likely come to expect that nothing is simple where FOSS licensing is concerned, and derivatives are certainly no exception.

For starters, where do we draw the line between what’s a derivative and what’s not? In the simplest case, if you take FOSS project A and modify it to add, remove, or alter functionality, you now have a new piece of software A' that is also obviously a derivative of the original A, as shown in the image that follows. This situation is fairly easy, though of course it relies on a judgment call of whether the modifications are substantial. Insubstantial modifications to A (such as fixing a typo in a comment) might not create the derivative work A'.

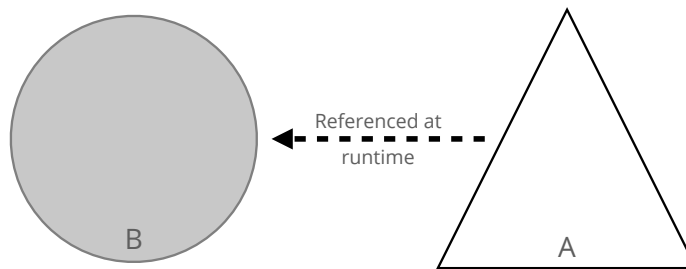


Furthermore, if you’re writing software B, and in its source code you include some or all of the source code from FOSS project A, it’s pretty obvious that since B derives functionality from A, B most likely is now a derivative of A, as shown in the following figure. This is also a fairly obvious judgment call.



Let's throw a spanner in the works. As mentioned in [Copyright, on page 24](#), creative expression falls under copyright but functional things like algorithms don't. Would it be possible to include the algorithms from A into B without B therefore being a derivative work? Technically, yes, this is possible, but it's very difficult to separate out the creative portions of software source code from the purely functional. A great deal of creativity can be involved when expressing an algorithm in source code. Do you still want to include that algorithm in your software? Perhaps not, since it now creates a tricky legal situation.

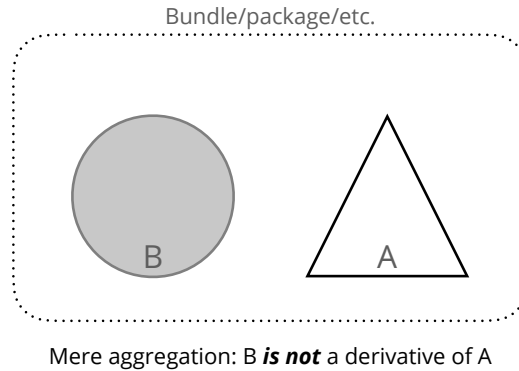
Now, what if your software B doesn't actively include (copy or *statically link* to) the functionality from A but instead only calls out to or refers (*dynamically links*) to it when you run software B? Is B still a derivative of A? It depends upon the license of A. For permissive or weak copyleft reciprocal licenses, no, B is not a derivative of A. The two are separate pieces of software that merely interact. However, if A has a strong copyleft reciprocal license, then, yes, the FSF considers B to be a derivative work of A. See the following figure.



Permissive or Weak Copyleft: B **is not** a derivative of A
 Strong Copyleft: B **is** a derivative of A

How about if you bundle A and B together in some way? Say you include them in the same zip file, tarball, container image, or other such packaging device? The preceding advice as far as combination and interaction still holds, of course, but if the two pieces of software don't otherwise interact and are simply traveling companions in the same bundle, then it's called a *mere aggregation* in IP legal circles. No derivatives are found in this situation, because the pieces of software aren't combined or interacting in any way. They're each there to perform their functionality entirely separately, as shown in the [figure on page 60](#).

To make things even more interesting (as though that were needed), we can't forget that this is a big world on which we live, and despite some standardization, copyright laws still can vary from jurisdiction to jurisdiction, including interpretations of what constitutes a derivative.



Keep in mind that whether a work is or is not a derivative tends only to matter when distribution enters the picture. If software B is something your company is developing purely for use inside of the company and it will never see the light of day outside of your organization, then B's derivative status doesn't matter. However, if the intention is to distribute B—even as an online service—then that status matters a lot since it can dictate the license under which you must distribute B.

Distribution

FOSS licenses are wonderful because they give anyone in the world permission to use, modify, contribute to, and share in the benefits of millions of software packages. If a package is under an OSI-approved license, your company may use it however needed to operate and innovate the software for their business. If the software your company creates is for internal use only, you don't even need to pay much attention to the licenses under which those FOSS packages are released beyond whether the licenses are OSI approved. But the moment your company distributes software, that distribution triggers the terms in every one of the FOSS licenses of the packages you used to create that software. Your company then needs to comply with those terms at the moment of software distribution, or it can get into a lot of legal hot water.

Distribution Defined

So what qualifies as distribution of the software? Basically, it's selling or giving away that software to anyone outside of your immediate company or organization. If the software your company has created is a derivative of any FOSS projects (you learned about derivatives in the previous section), and if your company then sells or gives away that software to customers, partners, or even to random strangers on the street, then it's distributed the software.

According to the FSF, even providing copies of the software to external consultants working for the company qualifies as distribution. If someone outside of your organization legally ends up with your company's software in their possession, then the software was distributed and your company must comply with the terms of any FOSS licenses.

Distribution also occurs if the software is embedded or running in hardware that your company sells or gives away. Every smart device in your home runs software to do all those fancy things like turn on your lights and notify you when your laundry is done. To sell those devices, their makers must also include the software to make them work. This therefore qualifies as distribution of software and triggers the license terms of any FOSS projects used to create that smart device software.

Distribution Over a Network

As you learned in [Network Reciprocal/Copyleft, on page 48](#), the Affero General Public License (AGPL) defines distribution slightly differently than any other FOSS license. Whereas most licenses are triggered by the basic possession type of distribution as previously defined, the AGPL triggers on that type of distribution and also if the software is available to others as a service over an internet network. To the authors of this license, using the software as a service is analogous to using the software on your own computer or other hardware and therefore also qualifies as distribution. So if your company operates its software as a service over a network and that software derives in some way from FOSS projects available under the AGPL, your company must comply with the terms of that license. As the AGPL is a strong copyleft/reciprocal license, this compliance includes making the source code for the service software available to users under the AGPL.

While it may appear moderately straightforward, there are of course a number of potential edge cases in this matter of distribution of software and how it relates to FOSS licenses. To be safe, check with your lawyer before providing software to anyone outside of your organization. Doing so will be cheaper than an unexpected license violation lawsuit and easier than an after-the-fact license compliance effort.

FOSS License Compatibility

As your company develops software, it's now certain that it will incorporate some open source in the mix. According to the 2021 Red Hat State of Enterprise Open Source report, 90 percent of IT leaders use open source in their

companies⁴ (and in my experience the other 10 percent do as well but are unaware of it). With so much open source in the mix, it's inevitable that it will include components released under several different free and open source licenses. Nothing's wrong with doing this, but if the software is going to be distributed at all, then that mix of licenses means you need to pay more attention to what they are. Not all FOSS licenses are compatible with each other.

License compatibility determines whether a combined work may be distributed and, if so, under which license that must happen. While every FOSS license provides the ability to run, study, modify, and redistribute the software to which it's applied, some licenses contain clauses that collide with and violate clauses in other FOSS licenses.

- Generally speaking, typically it's not a problem combining projects under different permissive licenses.
- Combining software under two different reciprocal licenses usually doesn't work unless one of the licenses has compatibility explicitly included in it.
- Combining software under a permissive license with that under a reciprocal is sometimes permitted and sometimes not.
 - For instance, the Free Software Foundation (FSF) considers the patent termination and indemnification portions of the Apache license to be incompatible with the GPL version 2 license; software using these two licenses can't legally be combined and distributed in any way.
 - However, the FSF went out of their way when drafting GPL version 3 so that there'd be no such compatibility concerns. As long as the resulting software is distributed under GPLv3 or later, FSF sees no problem with a combined work of Apache and GPLv3+ software.
- Speaking of multiple versions of a single license, as a general rule older versions of a license are compatible with newer, but the reverse isn't true. So you may combine a GPLv2 project with a GPLv3+, but the resulting distribution must be under GPLv3.

To make the process slightly less complicated, here's a license compatibility [matrix on page 63](#). If you have two FOSS components, A and B, this matrix can help you determine whether they can be combined in a work your company intends to distribute and, if they can, which license is most likely required for that distributed work. Keep in mind, this matrix is only a guideline and is not a replacement for professional legal advice. Before distributing a

4. <https://www.redhat.com/en/resources/state-of-enterprise-open-source-report-2021>

...with a project with this one & what is the likely license of the end product?

	MIT/X11	Apache v2	3-clause/ New BSD	2-clause/ Free BSD	MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
MIT/X11		any	any	any	MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
Apache v2	any		any	any	MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
3-clause/ New BSD	any	any		any	MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
2-clause/ Free BSD	any	any	any		MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
MPL v2.0	MPL v2.0	MPL v2.0	MPL v2.0	MPL v2.0		LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
LGPL v2.1	LGPL v2.1	GPL v2	LGPL v2.1	LGPL v2.1	LGPL v2.1		LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
LGPL v2.1+	LGPL v2.1+	GPL v2	LGPL v2.1+	LGPL v2.1+	LGPL v2.1+	LGPL v2.1		LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	GPL v3 or GPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+		GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3

Can I combine a project with this license...

work, always double-check with your lawyer to ensure you're in compliance with all of the component licenses.

Yes, this is as complicated as it sounds (and it gets even worse when a single work is under two licenses—or sometimes even more). Unfortunately, getting around that isn't possible, no matter whether the software is released under FOSS or proprietary licenses. The best you can do is be aware of the potential complication and be on the lookout for it.

As with most problems, the earlier you catch license compatibility problems, the easier they are to fix (typically by swapping out one of the components for another released under a different license). In [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#), you'll learn more about establishing processes to monitor your company's free and open source software supply chain. Among other benefits, these processes and resulting policies will help catch this sort of compatibility problem earlier in the software development life cycle.

Avoid Common FOSS Business Risks

So many companies and organizations are investing in their open source strategies and programs—and for good reason: doing so brings a host of benefits. You’ll learn about those benefits in [Chapter 5, Strengthen Your Business Through FOSS, on page 87](#), but first we need to get something out of the way. I won’t sugarcoat it: for all the benefits of a FOSS strategy, there are just as many risks.

Business is inherently risky; FOSS in business is no different. In both cases the risks are avoidable with education, intention, and attention. In this chapter you’ll receive the facts about those risks (education) without fearmongering. In future chapters you’ll learn not only how to avoid those risks but also how to convert them to benefits (with intention and attention). If you’re facing these risks right now and trying to figure out what to do about them, each risk section includes references to the parts of the book that will help you out of your jam.

Inbound FOSS Risks

These risks fall largely into the same categories as other topics in this book: inbound FOSS and outbound FOSS. Also like this book, you’ll learn about the inbound FOSS items first.

Supply Chain Awareness

When many companies start to think about FOSS with respect to their business, they immediately jump to the idea of releasing projects. This thought process often sounds like, “Let’s release this software! Everyone will show up and use it and love it and give us a lot of free work and advertising!” Other times, the thoughts are, “We have this software lying around. We don’t really use it anymore, so let’s release it as an open source project and let it be

someone else's problem. Then our company will get attention for releasing it." Recently though, the thoughts have trended toward, "We're building a company around this software. Let's give it away for free as a FOSS project, drive traffic to the website and gain name recognition, then figure out how to convert that into profit."

The Releasing-FOSS-Means-Profit stars in the eyes of these business leaders are blocking them from seeing the more obvious opportunities right in front of them. For the majority of businesses, the lowest hanging fruit where FOSS is concerned is not releasing FOSS projects but instead gaining awareness of the FOSS that they're *already using*.

As mentioned in [Chapter 1, Lay the Foundation, on page 3](#), if your company is building or using software, then it's already using free and open source components. You may not think you are, but trust me, FOSS is there. For instance, your software development team may use Git, a version control system released as Free Software. The company website may be based on WordPress, the most popular content management system in the world (and FOSS).

The majority of companies have no idea what sort of FOSS is in play within their organization. Teams need to get their jobs done, and FOSS enables them to do that more efficiently, so the teams use FOSS components then they move on to their next task. Meanwhile, these components continue chugging along in the background, unknown, untracked, and unnoticed—and potentially holding the door open for all sorts of unsavory and potentially devastating risks.

Therefore, before considering what software to release, companies should first look to see what software they're *using*. The almost complete lack of awareness of their FOSS software supply chain is a gaping hole in their overall strategy and can sink them as readily as a poor product launch.

Risks



- *Invisible links in software supply chain.* It's always difficult when a link breaks in your supply chain, but invisible links raise the stakes considerably.
 - *Compromised security.* Invisible links in the software supply chain don't receive security updates and are more susceptible to criminals.
-

Risks

- *Unknown license obligations.* The company is using software but is unaware of the obligations to which it has agreed by doing so.
- *Infringing creator copyright.* Using software without an explicit permission (license) to do so is illegal under copyright law.

Where to find help:

- [Chapter 11, Know the Links in Your Software Supply Chain, on page 173.](#)
-

License Compliance

Thanks to the attention it's received over the decades of free and open source software, when someone in a business hears the phrase "FOSS risks" they usually think about licenses and license compliance. This is largely due to the GPL/Copyleft/Reciprocal license fearmongering led by certain large corporations in the 1990s and early 2000s. Approaches such as calling Free Software "a cancer" caused many business leaders to create policies designed to restrict or prevent FOSS usage within their organizations. In many cases these policies were largely performative, as software developers in these companies found ways to use FOSS components anyway—often doing so on the sly—usually for pragmatic rather than nefarious reasons. While both the establishment and breaking of the policies may have been well-intentioned, both actions led to invisible links in these companies' software supply chains.

As you learned in [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#), software licensing is complicated. Usually, simply using the software reflects acceptance of the license and its terms and conditions. If you're not aware of what software you're using, you can't know what terms and conditions you've accepted. Your company may be making promises that it either cannot or is not prepared to keep.

On the other hand, the components in your company's software supply chain may not be licensed at all. A woeful number of seemingly open source projects are released without any license. Often this happens because the project maintainers are eager to share their work with the world but lack basic knowledge about copyright, licenses, and their importance. They don't realize that, legally, no one is allowed to use the work they've shared unless they've

given permission via a license. Software developers in your company share this lack of basic knowledge about copyright and licenses. Not having been trained to look for and be aware of the licenses on the software that they use, and being motivated to complete their assigned tasks, many will gladly use these “open” software components despite not having the legal right to do so.

Risks	
	• <i>Unknown license obligations.</i> The company is using software but is unaware of the obligations to which it has agreed by doing so. • <i>Infringing creator copyright.</i> Using software without an explicit permission (license) to do so is illegal under copyright law. • <i>Expensive legal fallout.</i> Lawsuits over infringed copyright or unmet license obligations can be expensive both in time and money. They also can damage your company’s brand. • <i>Expensive software rearchitecture.</i> Removing and rearchitecting around problematic software components can affect product availability and stability, leading to lower customer satisfaction. It also prevents software developers from using that time to fix other bugs and add new features.
	Where to find help:
	• Chapter 11, Know the Links in Your Software Supply Chain, on page 173 • Chapter 12, Maintain FOSS License Compliance, on page 201

Security

Gather ‘round, friends, and hear the sad tale of Equifax. Like pretty much all companies, Equifax uses FOSS in its business operations. In fact, its credit dispute website relied heavily on a FOSS project called Apache Struts. In early 2017 the Struts community located and fixed a major security vulnerability and told all project users to update to the latest version *immediately*. Many did. Equifax, alas, did not. Criminals used this Struts vulnerability to gain access to the Equifax systems. Over a series of months the criminals stole the private information of more than 150 million individuals, which is believed to be the largest data breach in history (so far). They were stopped when Equifax finally updated the version of Struts used in their software. While other factors contributed to the scope of this breach—for instance, the

internal systems weren't configured for isolation and prevention of cross-contamination—the fact remains that Equifax's failure to keep its FOSS supply chain up-to-date opened the door to an act that harmed millions of people.

It's a terrible tale of woe, and one that cost Equifax dearly, but unfortunately they're in good company. Criminals frequently exploit software security vulnerabilities to steal information and/or extort their victims. These vulnerabilities and security risks exist in proprietary software as much as in FOSS, but due to the prevalence of FOSS in software development, a decent percentage of these vulnerabilities turn up in FOSS components. The good news is that most free and open source projects prioritize rapidly locating and patching security. The bad news is that many companies, lacking awareness of their software supply chains, fail to notice this. An invisible or overlooked link in that supply chain may hide a ticking time bomb that can have a blast radius that includes your customers.

Some might try to take comfort in the fact that if they don't know what links are in their software supply chains then neither do the criminals, so of course their systems must be safe. This couldn't be further from the truth, as “security through obscurity” is little more than security theater. Criminals, you see, don't *need* to know what's in a company's software supply chain. All they need to do is run their exploit against as many companies' systems as possible and inevitably they'll find some that they can compromise. While criminals don't need to be aware of your software supply chain to break in and loot your systems, the only possible way for your company to prevent that looting is via supply chain awareness and maintenance. Otherwise you're one bad day from becoming the next Equifax.

This awareness and maintenance, by the way, includes software run in *containers*. To over-simplify it, a container is a self-contained snapshot of a running software system. This snapshot is easy to share, making it relatively simple to distribute software and even simpler to run it. Everything necessary for the software's operation is included in the container and correctly configured, so usually it “just works.” The most common methods for sharing and running containers are *images* and *files*. The image is an immutable and opaque snapshot; practically speaking you don't simply open this container and look inside to see what's inside it. The file is a recipe for building the snapshot; it's text-based and therefore less immutable and opaque. The problem with both methods is that often they don't receive either the necessary awareness or maintenance. Is that container image running an old and compromised version of a FOSS component? Was that container file updated to build with

the latest security patches applied? How do you know? All too often, the answer to that question is “you don’t.”

Risks



- *Invisible attack vectors.* Ignorance of the software in use leaves the company systems vulnerable to criminal activity.
- *Out-of-date software.* Neglecting software updates exposes the company (and its customers) to data breach via unpatched security holes.
- *Opaque containers.* Blindly using containers without knowing what software (and versions) they’re running increases the invisible links in the software supply chain.

Where to find help:

- [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#)
 - [Chapter 12, Maintain FOSS License Compliance, on page 201](#)
 - [The Container Complication, on page 184](#)
-

Liability

Following is an example of a Disclaimer of Warranty, excerpted from one of the most popular free and open source licenses. Basically, these disclaimers say that if you use the software and something goes wrong, then on your head be it. The creators of and contributors to the software wash their hands of the whole situation. You deal with it and bear the entire burden of correcting any problems, either express or implied.

Apache-2.0:

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

Such a disclaimer is standard fare in FOSS licenses, so much so that few if any of the most popular licenses lack one. The thing is—and this is implied

in the excerpted text above—these disclaimers aren’t valid in all jurisdictions (“applicable law”). If your jurisdiction isn’t one of those where the disclaimer is invalid, you may have little recourse should something go awry due to your use of the software.

In most cases, the worst-case scenario isn’t all that bad (relatively speaking). Perhaps your company will have to refresh its database from a backup because data was corrupted by a poorly written software component, or rearchitect its solution because a critical component contains a memory leak. These are inconvenient and perhaps expensive and disruptive to business operations, but are recoverable. It’s not like anyone died, after all.

But consider the example of a self-driving car that strikes and kills a pedestrian because of bugs in the machine learning and artificial intelligence components used in the self-driving software. Who’s liable for it? Is it the company that built the car? The one that wrote the self-driving software? The human copilot? The human who chose to use the buggy components? The authors of the components themselves? If those components are FOSS and use licenses with disclaimer of warranty clauses, are those authors protected from liability in this situation despite the disclaimer? This is a serious situation, after all; someone died because of that car.

This is obviously a matter for the courts to decide, and as they do so they’ll find surprisingly few cases for precedent. It’s not that the matter hasn’t come up, simply that it hasn’t especially been adjudicated. This leaves the matter of liability open to considerable interpretation.

Risks

- *Unaware of disclaimers of warranty.* If the company has invisible links in its software supply chain, it also may have invisible disclaimers of warranty in system-critical components.
- *Unclear extent of liability.* The variation in applicability of these disclaimers based on jurisdiction, as well as the relative lack of case law on the subject, leaves the matter of liability in a fog.



Where to find help:

- [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#)
 - [Chapter 12, Maintain FOSS License Compliance, on page 201](#)
-

License Changes: Relicensing to Non-OSI-approved

If you remember from [Chapter 2, Make the Intangible Profitable with Intellectual Property, on page 17](#), copyright holders get to say who has permission to do what with their works, and that permission takes the form of a license. Once they've chosen a license, a copyright holder may change their mind. It doesn't happen frequently, but FOSS components can be relicensed. The copyright holder (or, more commonly, holders) can release a version of the software that uses a totally different license. Everyone who uses earlier versions of the software can continue to use it under the terms and conditions of the previous license, but from that new version forward the new license applies to anyone who uses the software.

Relicensing like this isn't especially common because, frankly, it's a real pain in the butt for a FOSS project to do. It requires getting all of the copyright holders to agree that their contributions can be released under the new license, and for some projects that can mean tracking down hundreds or thousands of individuals. But for other projects, especially projects with a primary corporate sponsor or those with a contributor license agreement that includes copyright assignment or relicensing permission, there are usually considerably fewer copyright holders, so the overall relicensing process is much easier.

When a project relicenses, the new license doesn't necessarily have to be an open source (OSI-approved) one; it could be anything at all. A project under a permissive license like MIT could relicense to use GPL-3.0 instead. Recently the OpenSSL project released version 3.0, in which it relicensed from dual OpenSSL and SSLeay licenses to a single Apache-2.0 license (resolving a lot of license compliance headaches in the process). Projects could even relicense to a proprietary license with unknown and poorly understood terms and conditions.

For instance, in January 2021 Elastic announced that upcoming versions of their very popular Elasticsearch and Kibana projects would be relicensed from the open source Apache-2.0 license to the proprietary Server Side Public License (SSPL-1.0). This caught many companies that relied on Elasticsearch and Kibana off guard and was generally met with negativity from the FOSS community, but Elastic persisted in its planned license change.

Unfortunately, the latter situation is becoming more common for corporate-sponsored open source projects that form the core of a company with an *open core* business model. You'll learn more about these in [Chapter 22, Incorporate FOSS into Business Models, on page 369](#). Companies operating under this model give their software away for free under permissive FOSS licenses and then rely on converting those FOSS users into customers who pay for software

support or additional functionality. Many open core companies discover that they can't generate enough revenue with their business model and need to switch to a different one. Often this switch includes relicensing the previously open source software, something the company is able to do with ease if it hasn't encouraged or accepted many contributions from external entities, or if contributions required signing a contributor license agreement allowing them to make such changes. Typically the new license restricts what people are able to do with the software (such as launch their own product offering based on it) in order to protect the open core company's intellectual property and share of the market.

Now, consider a case where your company lacks visibility into its software supply chain and a link of that chain—such as Elasticsearch or Kibana—is a component that changes its license. Remember, use of the component equates to acceptance of its license terms, regardless of whether or not you're *aware* you're using the component.

In the world of copyright and licensing, ignorance isn't a good excuse for noncompliance, only a reason why you haven't complied *yet*. You have no way to tell when a license changes on a FOSS project if your company isn't aware of what the earlier license was in the first place, let alone what its obligations are under *either* of the licenses.

Risks

- *Unknown license obligations.* The company is using software but is unaware of the obligations to which it has agreed by doing so.
- *Expensive legal fallout.* Lawsuits over infringed copyright or unmet license obligations can be expensive both in time and money. They also can damage your company's brand.
- *Expensive software rearchitecture.* Removing and rearchitecting around problematic software components can affect product availability and stability, leading to lower customer satisfaction. It also prevents software developers from using that time to fix other bugs and add new features.



Where to find help:

- [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#)
 - [Chapter 12, Maintain FOSS License Compliance, on page 201](#)
-

Sustainability

Every company has a list of FOSS projects or components that the company truly relies on—the projects without which the company would be in some sort of trouble; which if the projects were to go away, the company would have to do a lot of expensive and inconvenient work to replace them; the ones which, if compromised, could potentially cause a great deal of harm to the business or to its customers.

Now—what’s the health of those projects? Are they maintained by one or two overworked and burnt-out developers? Are they maintained at all? Are their communities so toxic that no one wants to join and help support the project? In summary, are those projects sustainable?

The definition of “FOSS project health” is complex to the point that an entire organization is dedicated to working on it: the CHAOSS Project.¹ Many variables must be considered, and not all of them apply to every FOSS project or community. What *does* hold true for every FOSS project is that however you define it, an unhealthy project is an unsustainable project, and building your product, service, or company around unsustainable software is a recipe for business disaster.

Risks



- *Un- or under-maintained components.* FOSS components without adequate support cannot provide the security patches, bug fixes, or feature additions necessary to support active users.
- *Association with toxic cultures.* By using FOSS components created by unwelcoming and toxic cultures, your company tacitly accepts those behaviors and therefore risks damage to its brand and reputation.
- *Collapse of foundational components.* Unmaintained or unsustainable FOSS components may collapse, causing a fire-drill situation in your organization as it scrambles to find (or develop) and implement replacements.

Where to find help:

- [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#)
- [Chapter 13, Sustain. Maintain. Retain., on page 213](#)

1. <https://chaoss.community/>

Outbound FOSS Risks

Now that you have some idea of the potential risks of using (inbound) FOSS, let's learn about the risks your company could face by releasing or contributing to it (outbound FOSS).

Releasing open source projects is the new hotness for companies. A day doesn't pass when at least one press release or blog post announces yet another project pushed from the nest to spread its wings in the big world of open source. Similarly, many companies are starting to recognize the importance of FOSS to their businesses and that it's similarly important to contribute back in some way.

Unfortunately, where both releasing and contributing are concerned, relatively few of these companies think things through. They may have few to no goals for the effort, along with little process, policy, and follow-through. These companies either knowingly allow their software developers to do as they wish or, all too often, unknowingly do so. Other companies—especially larger enterprises—overthink in some ways (process and policy) but underthink in others (strategy).

Regardless of the approach, releasing and contributing to free and open source software projects hold a number of hidden risks for your organization.

Releasing IP

If you remember from [Software and Copyright: It's Complicated, on page 26](#), software falls into the intellectual property (IP) law bucket, typically either due to copyright or patents (or both!). Therefore, releasing and contributing to FOSS projects means sharing the company's (IP). Strategically speaking, is this the right move for your organization?

It could be that, yes, this is entirely the right move. Your company could get a stronger brand, easier recruiting, first-mover advantage, or any number of other strategic benefits from sharing its IP in this way. You'll learn all about these benefits in [Chapter 5, Strengthen Your Business Through FOSS, on page 87](#).

All the benefits are matched with just as many risks if the company allows its IP to leak in an uncontrolled manner. Corporate espionage is a thing: the secrets that these people steal can make them a lot of money and ruin the companies from which they steal. Unwittingly releasing company IP via FOSS projects is simply streamlining that secret-taking process, putting the information out there for all to see and saving those spies a lot of time and effort.

Without a plan and strategy for FOSS releasing and contribution, the company may release its “secret sauce”: intellectual property such as algorithms, features, and even products that differentiate them in the marketplace. Once this IP is out there under an OSI-approved FOSS license, competitors are legally allowed to use this IP in their own offerings. Releasing this IP can even enable new competitors to enter the market, building upon the IP that your company released. Any of these competitors, new or old, can use this IP to create a more compelling product offering than the one provided by your own company, degrading the company’s market share and revenue. Well, that’s capitalism for you.

Forethought, oversight, and strategy are critical where a company’s intellectual property is concerned. Without thinking it through, releasing IP under a FOSS license could potentially damage current or future strategic moves. Once the IP is loose in the world, it’s there for good. As the American saying goes, there’s no putting this toothpaste back in the tube. Depending upon the situation, a company may be able to relicense the IP under a proprietary license and/or take it down from the internet, but the damage may already be done. For however long the IP is available under a FOSS license, people can and will make copies of it. Changing the license later doesn’t prevent that nor does it make those copies invalid or illegal. There are nonprofits, such as Software Heritage,² that will archive all FOSS projects, saving them and making them available forever. As long as the software was under a FOSS license at the time of archiving, this is entirely legal, correct, and irreversible.

Risks



- *Leaking of differentiating intellectual property.* Strategically critical IP may unknowingly leak into the FOSS ecosystem and become available for any and all interested parties.
 - *Loss of market share.* Releasing company IP as FOSS can lead to new and powerful competitors who use that IP to launch or enhance their own product offerings.
 - *Limitation of strategic options.* Once the IP exists in the FOSS ecosystem, it exists there in perpetuity. If the release was unintended, then existing company strategy may no longer be valid.
-

2. <https://www.softwareheritage.org>

Risks

Where to find help:

- [Chapter 9, Establish Effective Inbound Policies and Procedures, on page 145](#)
 - [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#)
 - [Chapter 12, Maintain FOSS License Compliance, on page 201](#)
 - [Chapter 20, Releasing FOSS Projects, on page 317](#)
-

Security Considerations

No offense intended against you, but as clever as you are, I can almost guarantee that somewhere out there is a criminal who's even more clever. Some days the task of preventing these ne'er-do-wells from illegally accessing systems feels insurmountable. The baddies are always there, lying in wait and prepared to exploit every tiny gap in your company's security. Once inside, they'll either hold your systems hostage or simply steal the data and secrets outright then sell them to the highest bidder.

The acts of releasing and contributing to free and open source software projects can make it more difficult to keep the criminals outside your company's systems where they belong. All too often, sharing with the FOSS ecosystem ends up also sharing information that ought to be kept under lock and key, such as credentials for accessing those systems. Sometimes even exposed data—files that should remain private—are swept up into the contributions. This most often occurs in companies with little to no oversight or review of what sort of software or contributions the developers release into the FOSS ecosystem. The companies may not be watching, but the criminals most certainly are.

For instance, in 2020 the security researcher Craig Hays ran an experiment to see how long it would take criminals to take advantage of server access credentials that he intentionally published to the GitHub code-sharing platform.³ The server was what's known in security circles as a *honeypot*, set up specifically by a researcher to test for criminal activity. Once the server was available on the internet, it took only 68 seconds for a criminal to try to force their way in (unsuccessfully). However, it took a mere *34 minutes* for a criminal

3. <https://web.archive.org/web/20230607132522/https://craighays.com/what-happened-when-leaked-server-password-github/>

to locate the credentials that Hays had placed on GitHub, then use those credentials to gain complete access to the server. Had that been a real production server, containing customer data or proprietary secrets, it all would have been stolen or ransomed less than an hour after the login information became available on a public service.

Confidential information such as credentials is usually easy to spot before it's exposed and puts the company at risk—assuming a process is in place to do so. What's not so easy to see are the poor programming practices that can lead to releasing or contributing software that's vulnerable to compromise by clever criminals.

Industry best practices exist to help to minimize the risk of exposure,⁴ but not all programmers are aware of or use these. Preventing the release or contribution of vulnerable code therefore requires a process for review, but the review must include individuals who are skilled in spotting potentially vulnerable areas of the code. Without people with this knowledge, the chances of a criminal taking advantage of vulnerabilities in the code increase dramatically.

Even if your company has a policy and procedure that includes review of its own code prior to release to the FOSS ecosystem, there are hidden dangers in the FOSS code on which *your* code depends. These FOSS dependencies themselves may contain vulnerabilities that can allow criminals to compromise systems even if your own code is pristine and secure. Remember the sad tale of Equifax.

If your company contributes to or releases FOSS projects without first ensuring not only that all dependencies are secure but also that a process is in place to ensure those dependencies receive available updates quickly, then it's only a matter of time before a criminal locates those vulnerabilities and takes advantage of them.

	Risks
	• <i>Release of confidential information.</i> Lack of oversight or review increases the chances that login credentials or other confidential information leaks with FOSS contributions or project releases. • <i>Release of insecure software.</i> The contributions or software your company releases may not be developed according to industry best practices for ensuring security.

4. <https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/>

Risks

- *Compromise by way of unmaintained FOSS supply chain.*
Lack of awareness of the FOSS supply chain on which your company's contributions rely can leave open back doors that criminals can walk through unimpeded.

Where to find help:

- [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#)
 - [Chapter 20, Releasing FOSS Projects, on page 317](#)
-

Liability

Earlier in this chapter you learned about *disclaimer of warranty* clauses in FOSS licenses. You also learned that in some jurisdictions these disclaimers aren't entirely valid and that there's not a lot of case law about liability where FOSS components are concerned. Unless you're a skilled IP lawyer, the entire matter is about as clear as mud—and the situation isn't much clearer if you *are* a lawyer.

With that in mind, reconsider the previous example but from a different perspective: what if a self-driving car strikes and kills a pedestrian and free/open source software that your company either released or contributed to is a factor in the death because the software had a vulnerability that a criminal exploited to create chaos. Naturally you would feel horrible about it, as would everyone in your company, but feelings don't bring someone back to life. Neither do lawsuits, but that doesn't stop them from happening, so fingers can be pointed and blame laid. Will some of that blame end up with your company because of this FOSS component, and if so will it be held partially or wholly liable for the death of that poor pedestrian? Can you afford to risk it?

If the European Union has their way, the answer to that question will be no. The EU Cyber Resilience Act (CRA)⁵ would hold companies accountable for the security of their digital products, including FOSS components they create and release. With “fines of up to \$15 million (€15 million) or 2.5 percent of the offender's total worldwide annual turnover for the preceding financial year,” companies certainly could no longer afford to ignore vulnerabilities in their products.

5. https://en.wikipedia.org/wiki/Cyber_Resilience_Act

Risks

- *Unclear extent of liability.* The variation in applicability of these disclaimers based on jurisdiction, as well as the relative lack of case law on the subject, leaves the matter of liability in a fog.
- *Large penalties in new regulatory environment.* The CRA will change the liability landscape for companies and FOSS projects alike.



Where to find help:

- [Chapter 9, Establish Effective Inbound Policies and Procedures, on page 145](#)
 - [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#)
 - [Chapter 20, Releasing FOSS Projects, on page 317](#)
-

Trademark Considerations

Trademarks are an interesting facet of intellectual property law. Depending upon your jurisdiction, for a trademark to be valid it doesn't necessarily need to be registered, merely used and defended (though registered trademarks are easier to defend).

Your company undoubtedly has some trademarks. It may even have published policies about whether and how those trademarks can be used. Even though they're not companies, FOSS projects also can and do have trademarks. Many of them similarly have policies about use of those marks. When contributing to and especially when releasing FOSS projects, your company needs to be careful not to violate the trademarks of others.

For instance, if you develop and release a FOSS project that's a linter (automatic checker) for UX (user experience), it would be an unwise business decision to name that project LinUX. Doing so would be a quick and easy way to get the attention of the lawyers from the Linux Foundation, the multimillion-dollar nonprofit that manages the trademarks for the Free Software Linux operating system. Not paying attention to the trademarks of other companies and projects can lead to expensive and time-consuming legal actions and remediation.

Aside from potentially violating the trademarks of others, it's possible to complicate your own company's trademarks when releasing FOSS projects. All too often, companies release projects named either wholly or partially the same as the company itself. The typical thinking here is that the company's brand can receive a boost if the project shares the same name and receives attention, but that thinking ignores that not all attention is positive. Should the project be implicated in a security violation, develop a toxic culture, or any of a number of other unfortunate situations, then your company—having released, and sharing the same name as, the project—shares in the bad press. Instead of a boost, the brand receives a boot to the behind.

Name sharing between company and project can also impede community development for the FOSS project. When a project shares the same name as the company that released it, it implies a tight connection between the company and the project, which isn't always a good thing. To many potential contributors, this tight connection appears as though the company isn't acting in good faith, isn't looking to develop or listen to the community, and instead is using them as cheap skilled labor. Some people will still contribute to the project, but the fear of exploitation will keep many away.

Finally, what happens if your company releases a free/open source project that shares its name and then the company looks to be acquired? In a merger and acquisition (M&A), the strength of a company's trademark portfolio is part of its value. Depending upon the M&A situation, that FOSS project name may become a liability. Regardless, also at issue is what to do once the M&A closes. The new parent company will own the trademarks of the company that originated the FOSS project, but the status of the trademarks for the FOSS project can be less clear. Instead of dealing with these complications, some acquiring companies would rather shop around for an M&A with fewer legal question marks.

Risks



- *Violation of existing trademarks.* Inattentive use or release of FOSS software can violate the trademarks of other companies or projects.
 - *Dilution or damage to your company's trademarks.* Bad press or negative experiences with a project can damage the company brand and reduce the value of its trademarks.
-

Risks

- *Prevention of community growth.* Contributors are less likely to participate in projects where they feel their work may be exploited by the project's parent company.
- *Complication of merger/acquisition.* Company trademarks that become diluted or complicated via similarity to the trademarks of the company's FOSS projects can make it more difficult to close M&A deals.

Where to find help:

- [Trademarks, on page 19](#)
 - [Chapter 20, Releasing FOSS Projects, on page 317](#)
-

License Compliance

As your company prepares to release a project under a FOSS license, it's critical it be aware of the FOSS supply chain of dependencies of the project. Effectively no software is written without the assistance of free and open source components these days, and your project is no exception. Each component has its own FOSS license, and your company is required to comply with the terms and conditions of each one. Releasing a project counts as distribution of the software, and as you learned in [Distribution, on page 60](#), distribution is the trigger for those terms and conditions. If your company fails to comply with the terms of those licenses, it could be in for months or years of legal woes.

Similarly, it's possible that some of those links in the dependency software supply chain lack any license at all. According to copyright law, it's illegal even to *use* those dependencies—let alone distribute software that uses them—unless you have the express permission of the copyright holder(s) to do so. That permission is a license. No license? No permission. It's that simple. What isn't simple is the legal process that your company may face for releasing a project that violates someone's copyright.

The licenses of a project's dependencies can also impact the license under which your company must release the project. If the project includes dependencies that are licensed under reciprocal copyleft licenses, the reciprocity required by the terms of those licenses mean that your project must similarly be released under a copyleft license. Furthermore, your company must always ensure that people can access the project source code easily and that they have adequate scripts and documentation to allow them to build the software.

This is not inherently bad. In fact, as you'll see in [Chapter 19, The Strategy of Releasing FOSS, on page 297](#), releasing software under reciprocal licenses can be a powerful business strategy. However, if the internal business case for releasing the software is built upon the assumption that it will be released under a permissive license such as Apache-2.0, then discovering late in the process that there's a reciprocity requirement in a critical dependency can disrupt existing plans and strategies. Even worse, not discovering that requirement at all but launching the project anyway puts your company in violation of the terms of the reciprocal license and open to legal action. While it can be tedious, discovering and tracking a project's FOSS dependencies and supply chain is possible. Ignorance of a license or its terms is no excuse when faced with a violation lawsuit.

Risks

- *Violation of license terms.* Inattention to its FOSS supply chain puts the company at risk of violating the terms of licenses of components in that chain.
- *Violation of copyright.* Using someone's copyrighted work without permission is a violation and subject to legal action.
- *Disruption of business strategies.* Unexpectedly discovering a reciprocal requirement in a software supply chain can upset the best laid business plans of mice and men.



Where to find help:

- [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#)
- [Chapter 12, Maintain FOSS License Compliance, on page 201](#)
- [Chapter 20, Releasing FOSS Projects, on page 317](#)
- [FOSS License Compatibility, on page 61](#)

Maintenance Costs

People in free and open source software are fond of saying that there's *free as in beer* (no cost and enjoyable) and *free as in freedom* (as in the Four Freedoms of the Free Software movement). Both metaphors apply to free and open source software. Less frequently added to that "free" list but just as valid is *free as in puppy*. Free puppies are wonderful, with their uncontrollably wagging happy tails and their too-big-for-their-body paws and their face-licking

and their adorable little puppy snores as they nap. It's difficult to resist taking that puppy home with you, but if you do, then you need to care for it. It will need food, of course. And medical care, including getting spayed or neutered. And probably a dog license. And a collar and a leash and a bed and... No doubt about it, that free puppy is definitely not “free” in the end, but it's probably still well worth it. Take care of your puppy and you're rewarded with love. Neglect your puppy and you may get bitten (which you'd no doubt deserve) or land in court for animal cruelty.

When companies release projects as free/open source, they have metaphorically just acquired a free puppy, and it requires ongoing maintenance and upkeep to ensure it remains healthy and happy. Unfortunately and all too often, few companies consider the ongoing costs of maintenance when releasing a project. Instead, they do what we in the industry call *throw it over the wall*: they release the project and then more or less walk away. Sometimes there'll be a single internal programmer who dedicates some time to the project, but when that person inevitably moves on from the company, the project sits there, effectively abandoned.

As risks go, this isn't as dramatic as security breaches or license violations, but it still can cause considerable damage to a company's brand and reputation. Abandoned projects also waste the investment the company has made in the valuable intellectual property that they contain. In many cases, unless a company is prepared to plan for ongoing maintenance of a project they plan to release, it's less costly and more strategic not to release that project at all or to transfer the IP to an organization that has the resources to dedicate to maintaining the software.

Risks



- *Failure to plan for project maintenance.* Unmaintained projects degrade community trust in the company and damage the company's brand.
- *Waste of resources.* Abandoned projects waste not only the IP contained within them but also the time and effort required to release the project in the first place.

Where to find help:

- [Chapter 19, The Strategy of Releasing FOSS, on page 297](#)
 - [Chapter 20, Releasing FOSS Projects, on page 317](#)
-

Community Building and Management

In the movie *Field of Dreams*, Kevin Costner's character, Ray, is visited by a ghost who tells him, "If you build it, he will come." Based on this, he converts his corn field into a baseball field, resulting in a happy Hollywood ending for Ray and his family.

Many companies, as they release FOSS projects, are operating under what I call the Field of Dreams Fallacy. If you release it, they will not necessarily come. As with any endeavor, FOSS or not, a user base doesn't simply happen, nor does a vibrant community of contributors and supporters. Your company cannot release a project then sit back and wait for something to happen; it needs to take action.

More than code is needed for a robust FOSS project. Among other things, user and contributor bases require a lot of marketing and community management. Unfortunately, these are two skill sets that programmers often avoid or deride, so leaving these efforts to coders often means the work won't happen at all or won't happen well. As well, while there may be overlap between the user audience and the contributor one, they're still separate groups that require distinctly different approaches to foster and grow.

Both marketing and community management for a project require constant and ongoing attention if a company wishes to gain a return on their considerable investment in developing and releasing the project. While a press release at project release may gain some attention, without continuing outreach via relevant marketing channels, that attention will wane and no new attention will spring up to take its place. Without community management to guide that attention toward productive paths, very little of it will convert to contributions. And of course, a lack of community management can lead to the development of a toxic project culture that will reflect poorly on the company and its brand.

Risks



- *Lack of project visibility.* Without marketing, a project is unlikely to garner the attention required to fulfill the strategic goals for releasing it.
 - *Few external contributors.* Projects lacking in external contributors rarely or never gain the momentum expected or desired by the companies that release them.
-

Risks

- *Toxic project community.* Lack of community management allows the loudest voices to rule, and those voices rarely say things that reflect well upon the company or its brand.

Where to find help:

- [Chapter 20, Releasing FOSS Projects, on page 317](#)
.....
-

If you've gotten this far, you're probably pretty freaked out about this whole idea of free and open source software in a business context. In the wise words of the *Hitchhiker's Guide to the Galaxy*: *Don't panic!* Every single risk listed in this chapter is 100 percent preventable. This book is here to help you with that process. However, it was necessary to list these scary risks to help you spot them. After all, if you never see that snake that's hiding in the grass, you may step on it, and you're not going to like it in the least (and neither will the snake).

With all of these risks, why are so many companies relying on FOSS for their businesses? They're doing it because the benefits outnumber the risks and those benefits are compelling, as you'll learn in the next chapter.

Strengthen Your Business Through FOSS

[Chapter 4, Avoid Common FOSS Business Risks, on page 65](#), may have rattled you. It may be worrying to see potential risks all listed in one spot like that, but it's necessary. The only way to *avoid* a risk is to know that it is a risk. Without that knowledge, you're likely to follow the path of so many before you and blunder your company into a nasty situation. You won't know how you ended up there and, even worse, you won't know how to get out.

You may even be wondering why, with all of those potential and grim pitfalls, a business would get involved with free and open source software in the first place. It's important to remember here that, whether you know it or not, your company is already using and even contributing to FOSS. You're *already* involved, so wouldn't you like to make the most of it? And what does "make the most of it" even mean in this context?

In this chapter you'll learn that and more. You'll learn about the many benefits of FOSS to your business, not only how it saves you money but also how it can make you money. Please be aware, though, that these benefits don't come for free. The verb in "make the most of it" is active, and so must you be. The benefits listed in this chapter assume you and your company are taking the appropriate steps (or any steps at all) to gain those benefits. What those steps are—and how to take them—is the subject of the rest of the book.

Innovation and Digital Transformation

The business world is evolving and innovating faster than ever. New processes and technologies are bringing new levels of efficiency and opening new markets. Companies that don't evolve, that don't *digitally transform*, will be left behind. If you look at those new processes and technologies, you'll find they are underpinned by open source. FOSS is the whetstone of the cutting edge.

Without evolving your company's business processes and operations to take advantage of these new FOSS innovations, you'll fall behind—but your competitors may not. Digital transformation built upon a foundation of FOSS technologies can help your company remain nimble, scalable, and able to take advantage of future innovations.

Digital transformation isn't the only way FOSS contributes to your company's innovation. By using FOSS tools and technologies, your company can build better solutions faster.¹

In fact, you may find that your company is already benefiting in this way, as your development teams leverage FOSS components to launch more quickly. The FOSS ecosystem provides tried-and-true building blocks that your teams can assemble and integrate more rapidly than if they had to build everything themselves from scratch. A solution can go from idea to happy customers in a fraction of the time, or, in business speak, your solution will have a considerably faster time to market (TTM). If your competitors suffer from the all-too-common Not Invented Here (NIH) syndrome and believe they can build better components than they can get in the FOSS ecosystem, then your company will have a powerful advantage in the marketplace.

While using FOSS components is one way to stay on that cutting edge, participating in FOSS is another. Companies that are plugged into the FOSS ecosystem find it's a great way to learn about the Next Big Thing (NBT), often before it's gained its inevitable wide adoption. This foreknowledge provides your company the advance notice needed to capitalize on that innovation, gaining first-mover advantage in a new market.

Participating in these NBT projects does more than provide valuable market intelligence. As a good contributor and community citizen, your company is also in the position to influence development of the project in ways that benefit not only the greater community but also your internal strategies.

For example, companies that were early adopters of containerization and participation in the Open Container Initiative learned about and were able to test the Kubernetes open source container orchestration software before it became the Next Big Thing. These companies then formed the bedrock on which the popular cloud native movement and powerful Cloud Native Computing Foundation grew. Their early attention to FOSS innovations placed them in a position to leverage NBT technologies before their competitors.

1. <https://www.redhat.com/en/resources/state-of-enterprise-open-source-report-2022>

Benefits



- *Evolution to maintain and gain position in the marketplace.* FOSS is the foundation of digital transformation, enabling your company to compete more efficiently and effectively.
 - *Higher product quality and lower time to market.* By using proven and actively maintained FOSS components, your company can create robust solutions and provide customer value more rapidly.
 - *Gather valuable strategic insights.* When the most innovative solutions are FOSS, those who participate in the FOSS ecosystem will see opportunities on the horizon as they develop rather than after they've already passed them by.
-

Lower Total Cost of Ownership

In [Maintenance Costs, on page 83](#), you learned about the concept of *free as in puppy*. That adorable little slobber-factory that you picked up from the farmer down the road is only “free” in the sense that you lifted it out of a box and took it home; it still costs time and money to maintain. While people often apply this metaphor to FOSS, it’s valid for anything you create. You can save seeds or take cuttings for your “free” garden, but it still requires fertilizer, water, and weeding. You can release a product to solve customer pain points, but you still need to develop and release updates to keep the customers happy. The resources and effort required to maintain these is a large component of each thing’s Total Cost of Ownership (TCO). You can never avoid TCO altogether, but you and your company can do many things to reduce it.

Many hands make light work, and the FOSS ecosystem has many hands indeed. With FOSS, the TCO for any component is distributed across the entire community. The healthier the FOSS project, the wider that TCO spreads. Each contributor carries a smaller percent of the total burden, since everyone carries their share. You’ll learn more about FOSS project and community health in [Chapter 20, Releasing FOSS Projects, on page 317](#).

Consider this very basic and generic software stack for an online store:

• Operating system	• Firewall	• Security certificates
• Device drivers	• Database	• Business logic
• Network management	• Web server	• User interface

What would your company's TCO be if it had to build and maintain each of these components? Even as overly simplified as this example is, the TCO would likely bankrupt the company. Instead, you'll find that you can use a FOSS component for nearly every item in the list. Your developers can build the customized components like business logic and user interface out of FOSS elements such as JavaScript frameworks, so even these specialized parts end up costing less to acquire, build, and maintain. The TCO for this example online store can now be managed even by smaller companies, since multiple FOSS communities share the burden of building and maintaining it.

To give a sense of the value provided by FOSS components, in 2023 the Linux Foundation announced that if all of the FOSS projects under their oversight were proprietary software, it would cost USD \$26 billion in payroll to create and maintain it all.² These are only the projects housed in a single foundation. Millions of projects are either housed in other foundations or remain independent, so the overall payroll that would be required to develop and maintain all of FOSS would dwarf that \$26 billion.

Benefits



- *Download components rather than build for yourself.* The ability to choose pre-developed components removes the need to invest company resources to build them for yourself.
- *Lower maintenance burden.* When using FOSS components, the entire community—your company included—share in the maintenance burden so it impacts each community member less.

Reputation and Brand

The key to that lower TCO is that free and open source software is an ecosystem of sharing. People and companies come together to share their code and their expertise for the good of the entire community and beyond. Good FOSS citizens give back to the ecosystem at least as much as they take. Mechanisms such as reciprocal licenses help to ensure that this sharing occurs, discouraging a selfish “I got mine, everyone else can fend for themselves” attitude—and the poor community reputation that comes with it.

2. https://www.theregister.com/2023/09/28/kubecon_shanghai/

Do unto others as you would have them do unto you. Many of you (at least in the English-speaking world) will recognize this as the Golden Rule; it's a key concept in many religions and philosophies. What you've likely found by now, though, is that aside from it being the right and ethical thing to do, helping others has business value. When you give someone a gift, they feel well disposed toward you. When you donate to charities that you believe in, this in turn reflects well on you.

The same holds true for companies. Ethical behavior and sincere generosity is good for the company brand. Authentically good and helpful companies are pleasant to be around and to do business with, so others are more likely to seek them out and reward their supportive and welcoming behavior by giving them their business. This isn't *why* these companies behave in this way, but it's a valuable side effect: brand strengthening and awareness.

By participating in FOSS in an authentic manner as a good and responsible community citizen, your company gains the reputation of being generous, considerate, and community minded. Contributing back to and supporting the projects your company uses shows respect for the project creators and the overall ecosystem while also reducing the TCO for all who use the projects. Since all contributions are public, others are able to witness your company doing the right thing without your company necessarily crowing about it. The public nature of the contribution also helps to showcase the high quality of work that the company donates to the greater good of the community.

Fastly, a U.S.-based edge cloud and content delivery networking company, donates its services to a number of critical FOSS projects and communities on which it relies. As a result, these projects are more reliable and available for the entire ecosystem—Fastly included—while also demonstrating the capabilities and quality of those services. Fastly doesn't loudly advertise these donations itself, yet it still benefits greatly from the well-deserved reputation that its generosity has gained it. Fastly has a strong brand as a trusted and ethical company in the technology sector.

This sincere and public engagement with and support for FOSS components generates goodwill for your company, which is reflected in a stronger brand in the software developer market and beyond. It increases word-of-mouth marketing opportunities and, depending upon your company's product/service offerings, can lead both to filling the sales pipeline for a smoother overall sales process and, as you'll learn in the next section, to easier recruiting and onboarding.

Benefits



- *Increased visibility.* Public contributions, especially to high-profile projects, present your company in a good light.
 - *Improved reputation and brand.* Giving back to the community helps to earn the trust and respect of that community, strengthening the company brand.
 - *Faster sales cycles.* Companies with strong brands and good reputations have less trouble filling a sales pipeline and closing sales.
-

Recruiting and Onboarding

The reputation gained by being an authentic FOSS citizen helps even if your company's primary business isn't software or technology, since it undoubtedly employs software and technology people. While recruiting is a challenge for most roles, it's gotten especially difficult for technical roles in recent years. Despite all of the many ways people can learn how to program (traditional schooling, programming camps, and self-taught among others), there are still many more tech role openings than people to fill them. This has led to vicious competition to locate and hire these individuals, and eye-popping salaries to entice technologists to join.

Companies that participate in FOSS communities have a head start in the race to locate and hire technologists. These companies have found that community members make some of the best team members, as they're experienced in collaboration and communication across distributed teams. The open development model of FOSS projects means that your company can gauge someone's abilities via the technical and team leadership work they do in the community, making it easier to spot those who would be good candidates to join your firm. Once you've identified these people, your company's sincere and authentic participation in the project community will have earned it a reputation that will make working for it an enviable proposition for any community member.

Not only are FOSS communities rich pools for potential new team members, companies also find that the people they hire out of the community become more effective at their new jobs more quickly. When you hire someone from outside of the community, they may need a lot of time to get up to speed with your company's technology stack. This prolonged onboarding delays productivity and increases the cost of hiring. On the other hand, if your company

hires someone from the community of a FOSS component that's used in that stack, that new team member is more likely to be familiar with the technology and perhaps even with your business domain. This existing knowledge (plus familiarity with and to the community) allows them to make a difference quickly on their new team.

For example, if your company provides electronic components for automotive embedded systems, by participating in the Yocto embedded Linux project,³ it's likely to fill an open embedded system developer role more rapidly and with a more qualified candidate. If it creates Internet of Things (IoT) devices, then using, contributing to, and actively engaging with the Ditto digital twin project⁴ can ease the recruiting process for these highly sought-after developers.

Benefits



- *Ready pool of highly qualified technical candidates.* As a trusted member of a FOSS community, your company is in a position to locate and recruit skilled candidates it might not reach otherwise.
 - *Reduced onboarding times for increased productivity.* By hiring community members who are already familiar with the technology, your company reduces the training and time needed for new team members to get up to speed and become productive.
-

Staff Retention

As difficult as it is for companies to locate and hire new technical talent, many are starting to find that it's even more difficult to keep the people that they already have. Those eye-popping salaries that other companies offer are hard to resist, especially when coupled with bonuses and other benefits that your company may not be able to offer.

Staff retention is critical for business operations. It's expensive to recruit and train replacements, and when people leave they take their knowledge and experience with them. This is the sort of thing that can derail projects for months or more, compounding already expensive replacement costs that, according to Gallup,⁵ can range anywhere from 50 to 200 percent of the salary of the departing staff member.

3. <https://www.yoctoproject.org>

4. <https://eclipse.dev/ditto/>

5. <https://www.gallup.com/workplace/247391/fixable-problem-costs-businesses-trillion.aspx>

More often now, companies are turning to FOSS as one of the factors in retaining these valuable team members. Few companies participate in or support FOSS communities especially well, making it an appealing differentiator for those that do. This participation enables important career development for their staff members, who in turn receive enhanced personal and professional reputation by way of their FOSS community contributions. Those contributions can help to disrupt the monotony and boredom that can come with staying at one company for a long time, providing new challenges while still making a difference to company and team performance.

Savvy companies now realize that incoming technical staff have never known a world without free and open source software and expect not only that they'll be allowed to contribute to FOSS on company time but also that their employer will be an active and authentic member of strategically important FOSS communities. Technical staff members leave companies that don't meet these expectations and move to one that does.

Benefits



- *Career development for technical staff.* FOSS communities provide exposure to new technologies and the opportunity to learn and practice new skills that staff may not have the chance to learn at work.
 - *Retention of valuable staff members.* Supporting creative outlets such as FOSS community contribution can be the difference between a valuable staff member staying with your company or leaving for one that allows them more chance to contribute and grow.
-

Diversity

Recruiting and retaining excellent technical staff members via FOSS allows your company to build a more diverse (and therefore more effective) team.⁶ Most companies are relatively local in scope. While their products and services may be widely available, the company headquarters is in a single geographic area. Free and open source software, on the other hand, is global. Each project community can have members and participants from all over the world. This wealth of different perspectives is one of the reasons FOSS can out-innovate just about any commercial effort on the planet.

6. <https://hbr.org/2023/05/how-investing-in-dei-helps-companies-become-more-adaptable>

Aside from diversity, equity, and inclusion (DEI) being the right thing to do, studies show that it's also very good for a company's bottom line.⁷ A diversity of thoughts and experiences brings with it a diversity of ideas. It also brings the knowledge necessary to avoid pitfalls, leading to overall greater resilience. A diverse team is more effective at communication, reducing time to market and increasing quality. Diverse teams also are happier teams, improving staff retention.⁸ With all the proven benefits of building and growing a diverse workforce, it's incredible to find companies that ignore or merely pay lip service to the practice; but unfortunately, they're common. This means your company can develop and use its diverse workforce and culture as a differentiator to help it get ahead.

Some organizations, such as the following, are dedicated to boosting the diversity of FOSS and tech communities:

- *Outreachy*⁹ provides internships for people who are affected by systemic bias or are underrepresented in the tech sector in their regions, paying people to gain valuable technological experience by contributing to free and open source software projects.
- *Women in Linux*¹⁰ provides training and support for women looking to enter or level up in technology.
- *Lesbians Who Tech*¹¹ works to get more women, people of color, queer, and transgender people into technology.
- *Vets Who Code*¹² reskills veterans and their partners, using open source training and contributions to prepare them for technological roles outside of the military.
- *CODE2040*¹³ aims to improve racial equality in the tech sector, including free and open source software.

These are just a few of the not-for-profit/NGO institutions working to boost diversity in FOSS and technology. Company sponsorship of organizations such as these can help to increase the diversity of the overall technical workforce.

-
7. <https://insightglobal.com/blog/why-dei-is-important/>
 8. <https://learning.linkedin.com/resources/learning-culture/diversity-workplace-statistics-dei-importance>
 9. <https://www.outreachy.org>
 10. <https://www.womeninlinux.org>
 11. <https://lesbianswhotech.org/>
 12. <https://vetswhocode.io>
 13. <https://www.code2040.org/>

When your company contributes to and participates in FOSS community, it helps to build a culture of diversity, equity, and inclusion not only in the community but also among its own employees. Publicly fostering DEI in FOSS communities proves the company's dedication to diversity overall, including in its own ranks. This visible stand for DEI can attract diverse candidates and perspectives to your company's open job opportunities.

Benefits



- *Growth of diverse technical workforce.* Supporting organizations that promote and train diverse FOSS contributors increases the overall diversity of the tech sector.
 - *Identification of diverse candidates.* Participating in and fostering diverse FOSS communities leads to a more diverse pool of candidates, increasing the diversity of your company's employees.
-

Improved Security

In [Chapter 4, Avoid Common FOSS Business Risks, on page 65](#), you learned that FOSS can be a security *risk*, but it's one a company brings upon itself. FOSS is a security risk if a company doesn't pay attention to or maintain its FOSS supply chain. Rusted and corrupted links break (or are broken into), then everything that chain holds together comes crashing to the ground. On the other hand, a well-maintained FOSS supply chain is stronger and more secure than one composed solely of proprietary links. FOSS is among the most secure and trusted software options available.¹⁴

Yes, certainly, FOSS components contain bugs. Contrary to what anyone might lead you to believe, there's no such thing as bug-free software. And also yes, software bugs lead to software vulnerabilities and potentially to security breaches. Despite that, FOSS is still the superior choice from a security perspective. That's thanks to the free and open nature of the FOSS ecosystem and community. It creates an environment where bugs and vulnerabilities are located, fixed, and deployed more rapidly than is possible with most proprietary solutions.

For one thing, in our current technology world, FOSS components often have more users than proprietary software. More users means more people to discover and report vulnerabilities before they become problems or are exploited by criminals.

14. <https://www.redhat.com/en/enterprise-open-source-report/2022>

Next, the open and freely available source code ensures that anyone with the appropriate knowledge and an internet connection has the ability to inspect the code, regardless of whether they themselves use the component. A great many bugs and vulnerabilities are located simply because *someone bothered to look*. This isn't possible with the closed and private source code of proprietary solutions, since by their closed nature they prevent external stakeholders from inspecting the code.

Another reason for the heightened security of the FOSS ecosystem is the robust peer review that's required for many contributions to these components. Automated and manual tests and inspections occur before contributions ever reach the main source code, so dangerous bugs are spotted and squashed before they're ever released into the wild.

Finally, healthy FOSS communities include a diverse collection of highly and widely experienced software developers. This diversity of knowledge and experience provides far more bug- and vulnerability-spotting resources than your company could ever bring to bear on its own. All of these factors add up to locating and fixing vulnerabilities quickly and effectively.

The popularity of FOSS components also provides an element of increased security for the entire ecosystem. Because developers integrate and rely on these components to build millions of tools and services, a vulnerability in just one of those building blocks can affect a large amount of software. Thankfully though, it also means that fixes and improvements to each building block can telegraph to the software that includes it as a component. Fixing a vulnerability in one of those components then improves the security of every tool and service of which it's a part. The ripple effects of the free and open development processes and components are felt far and wide, leading to a FOSS ecosystem that's more secure than nearly any proprietary software.

Benefits



- *Fewer bugs and vulnerabilities.* The open community contribution process of FOSS enables rapid location of potential vulnerabilities, often even before reaching the software.
 - *Rapid patches of security holes.* Drawing upon a broad community of contributors, FOSS projects can more quickly mobilize to correct critical vulnerabilities as they're discovered in the wild.
 - *Access to skilled security professionals.* By working in an open community setting, your company has access to security professionals from across the industry.
-

Strategic Business Advantage

The preceding benefits apply to just about every company and, to be completely honest, are low-hanging fruit. While it does take persistent effort and some specialized knowledge, it's relatively easy to gain at least a little from each of those benefits with a modicum of intention and attention. Strategic business moves, on the other hand, are next-level stuff where FOSS is concerned and is where some real magic can happen.

As you'll learn in [Chapter 8, Inbound FOSS in Your Corporate Strategy, on page 131](#), and [Chapter 19, The Strategy of Releasing FOSS, on page 297](#), any number of ways are available for your company to use free and open source approaches to outfox your competition:

- Release and strongly promote a FOSS project to create a de facto industry standard.
- Engage in *coopetition*, releasing or contributing to projects that are critical to the competition to keep an eye on them.
- Develop a FOSS ecosystem around one of your company's services or product offerings, increasing the value to and demand from customers.
- Drive and shape industry standards early in their life cycle, giving your company a jump on their competition.

This approach isn't limited to your company's services or product offerings either. You could do the same FOSS ecosystem development around someone *else's* offerings, thereby attracting and capturing some of the existing market. You'll discover so many ways your company can get ahead by applying a free, open, and community-minded approach to its business.

Benefits



- *Cut off competitors' advantages.* Releasing projects for free and under reciprocal licenses can undermine competitors.
 - *Protect your own competitive advantage.* Strategic licensing of FOSS projects can ensure competitors must cooperate on your terms.
 - *Create new markets.* Development of FOSS ecosystems also develops new business opportunities. Creation of those ecosystems can give your company first-mover advantage.
-

As you can see, FOSS holds immense potential value for your company's business but only if you pay attention and manage it properly. The best way to do that is with an Open Source Program Office (OSPO), which you'll learn all about in the next chapter.

Introducing the Open Source Program Office

Now that it's buzz-worthy, more companies are realizing how critical free and open source software is or can be to their strategies and their businesses. Consequently, many of them are also realizing that they've been doing a very poor job managing their FOSS use and management as well as realizing exactly how much work (and how important) it can be to do so.

With any strategically important facet of company operations, a common approach is to create a team dedicated to it. FOSS is no different. More and more companies—such as household names like BMW and Amazon—are launching internal organizations with a mandate of managing the company's FOSS supply chain, community strategy, and engagement. This organization is the Open Source Program Office (OSPO). In this chapter you'll learn about the many roles an OSPO can fill in your company and the many roles it takes to fill the OSPO itself.

What Is an OSPO?

The primary characteristic of an OSPO is that it's difficult to define in a way that other team types are not. Sales sells things, Marketing promotes and builds brands, Product and Development create things, Finance handles the money, but what does an OSPO do? As you'll learn in this chapter, the answer is “it depends.” OSPOs can perform many different functions, depending upon the strategic needs of the company and its FOSS ecosystem. That doesn't mean we can't define it, only that it takes a little more effort. Thankfully, and according to the FOSS Way, we can get help from others in the community.

The OSPO Alliance is an organization for “promoting excellence in open source software management.”¹ Based in Europe, housed under the Eclipse Foundation, and founded by a collection of open source–related nonprofit organizations, the OSPO Alliance provides guidance and resources for companies wishing to improve the internal management and governance of their FOSS supply chains and related communities. The Alliance defines an OSPO as “...a cross-functional team to help define and steer an organization’s open source management strategy and organizational readiness.”

The TODO Group is another OSPO-related professional organization.² Housed under the Linux Foundation, the TODO Group is a community of practice composed of OSPO leaders from several dozen companies and universities from around the world. The TODO Group defines an OSPO as “...the center of competency for an organization’s open source operations and structure.”

Of the two definitions, I find the one from the OSPO Alliance to be complete and more accurate, primarily because it explicitly mentions the cross-functional nature of the OSPO. That cross-functionality is absolutely critical to OSPO success, regardless of the company or strategy. Regardless of the definition you prefer, OSPOs all share the end goal of ensuring that the company’s open source engagement supports its strategic initiatives and its business operations effectively and sustainably.

Does Your Company Need an OSPO?

Like most of the questions posed in this book, the first answer you’ll hear to this one is “it depends.”

Is your company in the business of shipping products or services that either are or depend upon software in any way? Then the final answer is “yes, absolutely.” At the bare minimum, your company requires a team dedicated to supply chain management and compliance, but it also requires someone to keep an eye on the health and sustainability of FOSS projects that are strategic to its business. Someone needs to be on the lookout for weak links in that chain and take action when they find one. Without an OSPO to do these things, your company is placing itself and its customers at risk.

If your company isn’t in the business of creating, distributing, or supporting products involving software, it still will *use* software. So it probably doesn’t need a full OSPO, but possibly can get away with having someone take on

1. <https://ospo-alliance.org>

2. <https://todogroup.org>

the role of maintaining the software supply chain as well as forming a cross-functional Open Source Review Board (OSRB) to help them review inbound FOSS for potential problems.

OSPOs in Academia and the Public Sector

This book is business focused. It's worth mentioning, though, that OSPOs are helpful beyond the corporate sector. In recent years we've started to see them popping up in all sorts of institutions, such as universities and municipalities.

- The city of Paris has invested a lot of thought and effort into its OSPO and is a great example of how FOSS supports much more than business operations.
- Universities from University of Santa Cruz in California to Trinity College Dublin and beyond have recognized that they have a duty not only to train tomorrow's FOSS leaders but also to maintain the ecosystem that enables world-changing research.
- Madrid created and then released Consul, a FOSS project for civic engagement and participatory government, which has been used by municipalities all over the world.

The OSPO++ community is the place to be to learn all about these noncorporate OSPO initiatives.^a

a. <https://ospoplusplus.org>

Functions of an OSPO

As previously mentioned, the functions that an OSPO performs depend entirely upon the strategic needs of the company. While it's possible for an OSPO to dedicate itself purely to a single role (compliance, for instance), this is rare. OSPOs almost always are cross-functional, and as such all OSPO team members require skills in communication and facilitation. In a single day the members of an OSPO may advise product development about how to release a utility as a publicly available free and open source project, revise marketing copy to more accurately describe a product based around a FOSS project, collaborate with security to manage patching a Free Software component that's updated to resolve a vulnerability, train presales architects on proper deployment of a FOSS toolchain, and any number of other interactions both internal and external to the company. OSPOs must be able to work across teams to accomplish strategic goals by performing some or all the following functions.

Coordinate

Cross-functionality requires coordination between and among different parties. The OSPO often becomes the hub of that coordination, tying together and leading the varied stakeholders. They do this internally in a company by coordinating between teams and units, externally by coordinating contributors and projects, and a combination of the two by coordinating internal and external stakeholders.

Collaborate

Open collaborations between varied stakeholders is critical to the success of any FOSS effort. The OSPO often drives these collaborations, be they externally on open projects and in open communities or internally by way of InnerSource, which you'll learn about shortly in [InnerSource: Collaboration Culture Comes to the Company, on page 107](#). Depending upon the ecosystem, these collaborations may even be between competitors in the marketplace (“coopetition”).

Community

Remember the coral from [Shared-Authorship Means More Copyrights Are Involved, on page 27](#)? Like polyps coming together to create a beautiful coral, people come together to create vibrant FOSS communities. Community is the superpower of FOSS, bringing disparate people together to create a whole greater than the sum of its parts.

Once a community exists, maintaining it as a healthy, welcoming, and supportive place is nearly as difficult as building it in the first place. These rules hold true no matter the type of community: around FOSS projects that the company has released, ones that the company uses, or those built internally to promote FOSS or support InnerSource initiatives.

OSPOs, when performing the community function, are agents of culture change. While sometimes communities are emergent—coming together on their own—more often, in a corporate setting building a community requires skill, intention, and attention. The members of the OSPO, with their deep experience with the FOSS ecosystem, have the skills necessary to build, strengthen, and maintain communities.

Comply

In the early days of free and open source software, what few nascent OSPOs existed were primarily concerned with matters of compliance—specifically, with matters of *license* compliance—usually in coordination with the legal team. More recently, FOSS compliance has become less of a focus and, especially in software startups, is frequently neglected altogether. Almost worse, it may become the pet project of one or two passionate individuals, Cassandra pleading against bringing Trojan horses into the company but ignored by programmers and leadership who are more concerned with moving fast and breaking things than with the safety of their products and customers. In companies where OSPOs perform a compliance function, they may create and deploy FOSS license compliance and supply chain security policies and procedures. They also monitor and administer to the company’s FOSS supply chain.

Promote

One of the most common mistakes companies make is to pay short shrift to the role of promotion in their FOSS strategies. This is an effect of the Field of Dreams Fallacy, mentioned in [Chapter 4, Avoid Common FOSS Business Risks, on page 65](#): if you build it, they will not necessarily come. OSPOs that operate in a vacuum don’t remain OSPOs for very long; instead they become the lowest hanging fruit when budget cuts roll around. To create momentum in and results from any initiative, it’s necessary to promote it, promote it, promote it, and then probably promote it again for the people in the back who missed it the first three times. For OSPOs, the promotion function comes into play in many ways, including internal FOSS evangelism—spreading the word internally and externally about the company’s FOSS projects and contributions, and on behalf of FOSS projects of strategic importance to the company to improve their organic sustainability.

Teach

You’ve probably noticed that doing FOSS well requires specialized knowledge and customized policies and procedures. If this information were obvious then you wouldn’t be sitting here reading this book. Because it’s *not* obvious, OSPOs often find themselves in the position of not only writing and maintaining those policies and procedures but also creating and delivering related

training materials. They also perform training and support for how to contribute to or become members of strategic FOSS projects and communities.

Develop

For some companies, it's strategically important to participate in and even influence the development and evolution of certain FOSS projects. Usually, these are projects around which the company has built a product offering, so it makes good business sense to invest in resources to ensure the projects remain active and healthy and that new features and releases can be anticipated in company product management. Commonly, this investment takes the form of assigning software developers to dedicate 100 percent of their working time to one or more of these strategic FOSS projects or developing and then releasing FOSS tools and projects on behalf of the company. In some companies these developers report into the engineering or product organizations, but it's common for them to be members of the OSPO. In this way they can bring their FOSS expertise to bear to support the overall FOSS program within the company.

Research

In a world where nearly every software advancement occurs first in free and open source software, few are better positioned or equipped to observe, research, and report on new developments than the OSPO. With boots and ears on the ground, OSPO members can discover new and upcoming technologies of potential strategic value, learn of shifts in the marketplace and ecosystem, and gather user needs and requirements. Having an OSPO perform the function of information gathering can provide your company the data and time it needs to make shifts in strategy before its competitors.

Strategize

Companies that look to the OSPO to provide data to inform strategies often also rely on the OSPO to develop those strategies. It can make sense to push this type of high-level decision-making as close as possible to the people who can implement it. This strategic function can include tasks like determining which projects, communities, and foundations are important and how best to engage with them to achieve company goals. Also included are gauging the health of strategic FOSS projects and communities and then deciding which actions are needed for improving and/or maintaining that health.

InnerSource: Collaboration Culture Comes to the Company

Free and open source communities create software of astonishing quality, despite the individual developers rarely (or never) meeting in person. They're able to do this thanks to the decades of experimentation by those who came before them, leading to best practices for FOSS development. You can learn about these best practices in *Forge Your Future with Open Source*.^a

Seeing the success of open source development models, some companies apply those best practices internally using a concept known as InnerSource. Teams across the entire organization can collaborate on each other's projects, sharing insights and experience. By breaking down the silos that usually isolate different teams, they're able to communicate and work more efficiently.

While in spirit InnerSource is closely related to open source, in practice it requires a different approach; therefore, I won't cover it in this book, but you can learn all about it directly from the helpful, friendly, and enthusiastic InnerSource community.^b

a. <https://fossforge.com>

b. <https://innersourcecommons.org>

What Kind of OSPO Do You Need?

As mentioned previously, no two OSPOs are alike in composition or functionality. They do, however, tend to fall into a set number of categories or types. Each type of OSPO will share characteristics with and perform some of the duties of other types, but each also tends to focus on one specific area. In this section, you'll learn about the most common types of OSPOs to help you determine which kind best fits your business needs. Note that these types are guidelines, not rules. It's possible (likely) for an OSPO not to fall neatly into one category or another. This isn't, after all, an exhaustive list of OSPO types, only the ones most commonly found in companies. If your company finds that a different type of OSPO than one listed here will best suit its needs, then by all means create it. However, you'll likely find that this section will help you decide what type of OSPO will be best for your company.

Compliance

In [Comply, on page 105](#), you learned that the earliest OSPOs were focused on license compliance. Because compliance was a focus in the early days of the business of FOSS, the oldest and most common OSPOs have license compliance as their primary concern. So you'll also find that some of the most senior

and experienced OSPO professionals, having cut their teeth in this early environment, similarly will have a strong focus on license compliance and the FOSS software supply chain.

Compliance-type OSPOs characteristically collaborate very closely with the company legal and compliance teams, especially around issues of license compliance, patents, and intellectual property protection. They'll try to keep a close eye on the company's inbound FOSS software supply chain and outbound FOSS project releases to ensure compliance with their license obligations, especially where reciprocal FOSS licenses are concerned.

More often recently, thanks to their existing knowledge of and attention to the supply chain, this type of OSPO also collaborates with the company's IT security team to help identify and remediate any compromised links in the FOSS supply chain before they can cause damage to the company, its operations, and especially its customers.

While it's most common for their attention to land on FOSS licensing, in companies that function in highly regulated industries like financial services or healthcare, this type of OSPO also assists the company to keep their use and release of FOSS components on the right side of government regulations.

The compliance-type OSPO can exist in any type of company but is most often found in companies where risk mitigation is a business operations motivator. This includes large, well-established enterprises, companies providing a product offering with a hardware component (for example, automotive components or networking devices), or risk-averse industries such as financial services.

The advantage of a compliance-type OSPO is that it typically will spot potential risks—license compliance or security—before they become problems. Especially for hardware companies, where the component ships with software that may not be easy to upgrade, this type of OSPO can help avoid costly license compliance missteps. In companies that rely on their patent portfolio for a significant revenue stream, the oversight provided by compliance-type OSPOs can better ensure growth of that portfolio.

The disadvantage of a compliance-type OSPO is it's very inwardly focused. As such, it has little to no involvement with FOSS communities—strategic or otherwise—and is otherwise not especially visible external to the company. Inside the company, the policies and procedures required to keep that close eye on the supply chain are seen by many (especially software developers) to be a burden to work around rather than a critical risk-mitigating business function. This can lead to an incomplete view of what FOSS the company uses, making it nearly impossible to correct security or licensing vulnerabilities.

Community Focus

Community focus-type OSPOs devote the majority of their efforts toward building community both internal and external to the company. They identify strategic FOSS projects, then work to ensure their sustainability by building larger and healthier communities and increasing contributions both from their own company and from other companies and individuals. If the company has released their own FOSS projects, the OSPO will monitor and grow those communities as well if it's of strategic value to the company. This growth requires increasing the diversity, equity, and inclusion of the communities, ensuring that they're welcoming to and supportive of contributors and their efforts. This includes putting in place and empathetically enforcing strong codes of conduct in the communities, making sure that everyone knows what is and isn't "how we do things here." You'll learn more about this in [Chapter 20, Releasing FOSS Projects, on page 317](#).

This type of OSPO most commonly arises in early- to mid-stage software startups, where its brand- and reputation-building is seen as valuable word-of-mouth marketing when a company may not otherwise be able to afford a large marketing budget. It's also often found in companies that are aware of and value the sustainability of strategic FOSS projects.

The advantages of a community focus-type OSPO are that it can develop more authentic and trusted company brand in the targeted strategic communities. The OSPO is a valuable resource for identifying and recruiting software developer candidates. With its proximity to the projects and communities, this OSPO also can provide early identification of potential weakness in these strategically important links in the company's FOSS supply chain, allowing adequate time for correction (or, in more drastic cases, software replacement and rearchitecture).

That community focus-type OSPOs provide business value but find it difficult to measure or prove this ROI is one of the common disadvantages of this type of OSPO. Internally, it therefore can be viewed as a nice-to-have but not as critical. Because building trust and communities is an investment that takes time to pay off, this OSPO frequently finds itself on the chopping block when it can't show immediate results rather than being afforded the time necessary to complete the long-term investment. The intimate and embedded nature of the roles required to build and maintain communities requires attention and careful management to avoid the high level of burnout prevalent among team members in community focus-type OSPOs.

Marketing

Marketing-type OSPOs are all about outreach, spreading the word of the good work the company is doing with, in, and for FOSS. For some companies, the developer relations/advocacy (DevRel) team will report into the OSPO if it's a marketing type. Even if the direct reporting structure isn't there, close collaboration will exist between the OSPO and the DevRel teams. The outreach-centric mission of this type of OSPO leads it to participate in many industry conferences and events, both by sponsoring and maintaining a booth at the event and by delivering presentations there. Other common forms of outreach for marketing-type OSPOs include articles, podcasts, and webinars.

Late-stage software startups most commonly feature the marketing-type OSPO, especially those with product offerings built around free and open source projects. These companies are in the *open core* category. You'll learn more about this in [Chapter 22, Incorporate FOSS into Business Models, on page 369](#). In these cases, the OSPO provides additional marketing and branding to support the growing sales efforts of companies that are nearing IPO or some other liquidity event.

Depending upon the company's target markets, a marketing-type OSPO can be a better source for outreach than traditional marketing efforts. For instance, it can speak more effectively to software developers, DevOps, security specialists, and those in other highly technical roles as well as to members of the relevant FOSS communities. By diverging from the standard methods of enterprise marketing, a marketing-type OSPO has more alternatives for company differentiation in methods of outreach. This can lead to more effective outreach and potentially a higher ROI than traditional forms of marketing.

Those methods, however, don't come cheap. Conference sponsorship and travel is very expensive, so marketing-type OSPOs require adequate financial support to be successful. Like all marketing, it can be difficult for this type of OSPO to measure and prove the value it's providing for all that money. Those efforts—expensive or not—may also backfire. For good or ill, a large percentage of people in highly technical roles distrust anything that resembles traditional marketing and may view the OSPO's outreach as insincere or inauthentic, providing a thin veneer of “open washing” over a standard corporate offering. The members of this type of OSPO must tread carefully and always work to maintain authenticity with all of their outreach efforts to avoid alienating the very people to whom they're trying to speak.

Product Focus

Product focus-type OSPOs spend most of their time in and around projects released by the company and/or used in its product offerings. As such, this OSPO's attention is often (but not always) spent on project features and feedback rather than community, though for projects released by the company the OSPO may build and support their communities. It's more common, though, for the company to be the primary or only contributor to these projects (*single-vendor commercial open source projects*) and for the community members to be project users rather than contributors. With this sort of project and community configuration, product focus-type OSPOs frequently include developer relations/advocacy specialists in the team.

This type of OSPO exists most often in open core software companies formed around single-vendor commercial open source projects (see [Chapter 22, Incorporate FOSS into Business Models, on page 369](#)). By deploying DevRel professionals to train and gather feedback from the user community, the OSPO is able to gather market research that can evolve the product in strategic directions while also generating sales leads via outreach.

Having staff members dedicated to and embedded in the projects in this way, product focus-type OSPOs can provide high-quality product feedback and market research for the company's product offerings. They can enable a strategic nimbleness by seeing and reacting to trends in the marketplace before competitors. The direct connection to the user community acts as a highly effective and efficient channel for marketing directly to the people for whom the message will be most relevant. It also allows the team to spot new additions to the community and nurture these relationships in a hands-on way to provide supremely qualified sales leads.

The product focus of this type of OSPO, as well as the single-vendor project approach that's common in these situations, can lead to friction between the company and a community that may be expecting a more collaborative project experience. Balancing the needs of the company product roadmap with the functionality expectations and desires of the community can alienate the community members by looking like inauthentic FOSS engagement, with the OSPO appearing to prefer sales pitches to collaboration.

Development-centric

Development-centric type OSPOs tend to focus on sustaining and improving FOSS projects that are strategically important to the company. By making this sort of investment to strengthen critical links in their software supply

chain and influence the roadmap of the projects, the company similarly strengthens the foundation on which their most important product offerings are built. While this OSPO team may include a relatively high percentage of software developers, these people will be embedded directly in and spend all of their work time with those external FOSS communities, making few direct contributions to internal initiatives or products.

Companies with product offerings built around large FOSS projects are most likely to have a development-centric type of OSPO. These companies have recognized the strategic importance not only of ensuring the health and sustainability of these projects but also of influencing the direction of project development.

The actions of development-centric type OSPOs ensure project stability, enabling the company to make long-term decisions and strategic plans. By encouraging OSPO members to earn leadership positions within the FOSS projects, the company can more easily direct the project toward developing feature sets of importance to company product goals, and when the roadmap heads in a direction less conducive to those goals, the company can react and change their own direction in plenty of time. Investing in projects in this way and engaging with the community in an authentic and supportive manner can lead to a strong company brand within the community, easing the hiring and recruitment process.

However, it's also common for development-centric type OSPO team members who are thus embedded in their respective communities to have a difficult time balancing their allegiance to the company with that to the project. Frequently they become isolated from the rest of the company, leading them to identify more closely with the community than with the employer paying them to be there. Sometimes they do a poor job of authentic engagement, such that attempts to influence the project roadmap present instead as attempts to control. This can alienate a community and degrade or even eliminate trust in the company. Being isolated from the rest of the company, it can be difficult for team members in this OSPO to set appropriate feature expectations within the company's product organization. This lack of communication and related understanding can lead to friction between product and project, placing the development-centric type OSPO in the middle and ill-equipped to manage the conflict.

Think Tank

Think tank-type OSPOs are not common but deserve mention. This type of OSPO typically participates in FOSS projects and communities at a high level

and are primarily observatory or influential, with relatively little focus on external community building. It will have more of a 50,000-foot industry- or ecosystem-level focus, keeping an eye on how things develop and looking for trends. Members of this OSPO frequently present their findings at industry conferences and in articles written for public consumption, while developing internal strategy-positioning documentation. This OSPO usually will also work to maintain and support FOSS participation across the company, but these efforts may take a back seat to their other research and influence duties.

The think tank-type OSPO rarely exists in product companies but can be a good fit for consulting or analyst firms, since their main deliverables include research and information gathering to inform strategies (either for the company or its clients). This research leads to “thought leadership” opportunities that, if properly directed, can establish or strengthen the company brand.

Companies that rely on the most relevant and up-to-date market research can benefit from a think tank-type OSPO and the influence it can bring to bear in strategic ecosystems. This OSPO is forward-thinking and can be the source of ingenious and industry-leading product development plans, positioning the company as highly innovative and cutting edge.

Due to their high-level engagement with projects and their ecosystems, the think tank-type OSPO can present to the community as overly corporate, cutthroat, and insincere. This can potentially damage the company brand. The nature of their duties can lead members of this OSPO to be more isolated from the rest of the company, leading to it becoming an underused and underappreciated resource.

Roles Inside an OSPO

In this section you’ll learn about some of the roles most commonly found in OSPOs, but remember—not all OSPOs will require people to fill all of these roles, and your OSPO may require roles that aren’t listed here. The roles represented will vary based on the type of OSPO your company has as well as the functions that the team performs. Also remember that it’s possible for a single person to hold multiple roles. This can be OK, as the team or role is just getting started, but I caution you not to maintain this arrangement for too long. Not only is it a recipe for burnout for that person, it’s also a signal that the company may not be adequately invested in the success of their FOSS initiatives.

The size and composition of any team in your company depend on several factors. Budget is a consideration, of course. This is typically the most obvious

constraint and enabler in all questions of staffing. Other factors include the acknowledged strategic importance of the team to the company, the maturity of the initiative, and the availability of people to fill the desired headcount. The OSPO is no different, so you should be at least moderately familiar with these staffing challenges.

If your company is just starting out on its strategic FOSS journey, I urge you to determine that strategy and the type of OSPO necessary to support it *before* you start the hiring process. This may require professional assistance, either in the form of an experienced and trustworthy consultant or by hiring a similarly experienced person as the OSPO leader tasked with shaping both the strategy and the team. Regardless, the strategy must come before the expensive (both in time and money) process of recruiting and hiring. Once the strategy is more or less set, sketch out the shape of the team and the roles necessary to meet the established goals. Invest in success! Shortchanging the team sets it and the strategy up for failure. If your company can't afford to hire staff to meet the stated strategic goals, those goals are unrealistically ambitious and need reconsideration. Remember, success doesn't happen overnight. It's OK and even advisable to plan the staffing investments out across several years in a staged manner.

Leadership (Head of OSPO, Chief Open Source Officer)

Most commonly, the leadership role has the title of head of OSPO, but companies for which FOSS is vital and strategic for business operations should consider acknowledging the cross-organization importance of FOSS by creating a top-level chief open source officer role that reports directly to the CEO. Whatever the title, this role leads the strategy, tactics, and team required to deliver on the company's FOSS initiatives. A person who takes this role must have considerable knowledge and experience both in FOSS and in business as well as solid communication, collaboration, political, strategic, and management skills. This role most often interacts with the OSPO team, company leadership, and external organizations and partners.

Supply Chain Manager

The role of supply chain manager is responsible for FOSS supply chain administration and maintenance. Their role may also encompass the full software supply chain, including proprietary and vendor-provided software. A person who takes this role requires strong knowledge of FOSS licensing, software security, software deployment mechanisms, the software development life cycle at your company, and software composition analysis tools. While

it's not always required, it can be helpful for a person in this role to have knowledge of programming or scripting. This role most often interacts with the legal, security, and software development teams.

OSPO Tooling Developer

Software supply chain tooling doesn't "just happen." Someone needs to build it. Even if that building is limited to integrating tools from different vendors or FOSS projects, it still requires a lot of work to assemble and maintain. This person also looks after any other tooling that the OSPO requires, such as community metrics dashboards and administration of public code forge accounts like GitHub or GitLab. This role most often interacts with the legal, security, and software development teams.

Technical Marketer

Companies often overlook the role of technical marketer when developing their OSPO staffing plans, much to their disadvantage. No matter the type of OSPO, technical marketing is critical for enabling and supporting efficient communication. This role is responsible for developing and delivering all internal and external outreach related to FOSS matters. A person who takes this role requires knowledge of and experience with FOSS in some way (ideally but not necessarily as a contributor of some sort), experience with community development, and of course experience with marketing operations and channels. Naturally, this role requires excellent communication, writing, and presentation skills. It most often interacts with the company marketing department, software development, and product management.

Project/Program Manager

Be it a single project or an entire program, someone needs to ensure that all the ducks stay in a row. That someone has the role of project or program manager.

People in this role must be exceptionally well organized with superb communication, conflict resolution, and project management skills and experience. This role most often interacts with FOSS projects, communities, and foundations as well as with other internal and external stakeholders for the project or program. For an excellent guide on program management for open source, see Ben Cotton's book *Program Management for Open Source Projects: How to Guide Your Community-Driven, Open Source Project*.³

3. <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>

Software Developer

In the context of an OSPO, the role of software developer contributes directly to the FOSS projects that the company finds strategically important. These contributions most often are code but can also be design, architecture, documentation, security reviews, or any other operation required for the technical success of the FOSS projects. The software developer role naturally requires adequate technical experience in the programming language/environment/ecosystem of the strategic FOSS project(s), as well as good communication skills. It's useful if a person who takes this role has previously contributed to FOSS projects or communities, but it's not required. This role most often interacts with the FOSS projects and communities, internal software development teams, and product management.

Technical Writer

Despite studies consistently and reliably showing that abundant quality documentation is key to the success and sustainability of FOSS projects,⁴ relatively few companies acknowledge the importance of documentation by filling technical writer roles in their OSPO. This is a tragic loss of the opportunity to make a difference to the ecosystems of strategic FOSS projects.

A single dedicated technical writer can be a powerful force multiplier for your company's FOSS initiatives. Someone who accepts this role will write and edit all types of documentation, tutorials, articles, and other content related to the OSPO, its programs, and strategic FOSS projects. They will require exceptional technical writing and editing skills and experience, as well as sufficient knowledge of the software development life cycle and the domain of the OSPO's strategic initiatives and FOSS projects. It can be helpful if they've previously contributed to FOSS projects and communities, but it's not required. This role most often interacts with marketing, software development teams, supply chain management, product management, and the FOSS projects, communities, and foundations with which the company is involved.

Community Manager

The role of community manager is to maintain, develop, and sustain the communities related to the OSPO and its programs. These communities may be internal or external to the company. Individuals who take on this role must have excellent communication skills, experience building and managing distributed and diverse communities, and experience contributing to or

4. <https://opensourcesurvey.org/2017/>

interacting with FOSS projects and communities. This role most often interacts with FOSS projects, communities, and foundations, and the company's internal communities, project/program management, marketing, and software development teams.

Developer Relations/Advocate/Evangelist

This role has several titles—developer relations, developer advocate, and developer evangelist. Each of the titles represents a slightly different set of responsibilities but will be treated the same here. Basically, where the OSPO is concerned, the role promotes and supports OSPO projects and programs by answering questions, providing training, and collecting and distributing feedback from users and communities. Individuals in this role require excellent communication, writing, and presentation skills and sufficient technical skills and experience for their assigned projects and communities. This role most often interacts with the FOSS projects, communities, and foundations of interest to the company, product management, marketing, software development, and current and potential customers of the company.

Code Forge?

In the previous section [OSPO Tooling Developer, on page 115](#), I mentioned *code forges*. If you're not in FOSS or software, you might not be familiar with this term. Let's fix that!

A forge is where people come together to access and collaborate on a free and open source project. Typically, forges are based around a Version Control System (VCS) and include other useful project management features like issue tracking, release management, and running test suites.

Forges are more than functional. They're also the home base and primary source of truth for a FOSS project and community. It's where you can always find the most recent version of a project, even if that version hasn't been released yet. It's also usually where you can find the canonical documentation and governance.

You'll learn more about forges in [Selecting a Collaboration Forge, on page 341](#), when we talk about releasing FOSS.

Placement of an OSPO Within an Org Chart

When forming an OSPO, a common mistake is to place it anywhere that has budget or space within the org chart rather than where the team can be most effective, showing an unfortunately typical lack of strategic insight and thinking on the part of the company. Sometimes, rather than intentionally forming an OSPO, the company instead accretes one—like a pearl forming

around an irritant in an oyster—slowly growing around a single individual so passionate about FOSS that they’ll do anything to get the company to do more with it. In these cases, like a pearl, the OSPO will sit in the dark where it forms until or unless someone moves it somewhere it can shine.

Handled correctly, the placement of the OSPO within the org chart depends upon the type of OSPO and its strategic remit. For example, compliance, development-centric, and think tank OSPOs typically are most effective under software/hardware development leadership, such as the CTO or a VP of engineering. Community focus OSPOs can be effective in the same location or under marketing. This is, naturally, where the marketing type OSPO will shine, while a product-focus OSPO is likely to be most successful within the product organization. When you read it laid out like this it all seems too obvious, doesn’t it? Yet despite that, OSPOs still get stuffed in whatever organizational cubby is most convenient at the time.

Occasionally, you’ll find OSPOs that are federated or matrixed across an organization, with a team here, a team there, all with different reporting chains. While it’s possible for this sort of decentralized OSPO to work, it’s a lot less likely. The differing reporting chains create a great deal of political friction, and they make it a challenge to create and execute a coherent FOSS vision and strategy. Typically, it’s better to have a single source of truth where your company’s FOSS endeavors are concerned.

Regardless of where the company strategically places an OSPO, it must report to a high-level leader to set it up for success. OSPOs must work across the entire organization, leveraging influence and collaboration to deliver on their remit. If this team reports to someone lower in the hierarchy (depending on the size of the company, anywhere from a director on down), it’s unlikely to have the clout and authority necessary to effect change when influence and collaboration fails them. Reporting to a high-level leader also shows to the rest of the company how important the company views the operations of the OSPO and acknowledges the skills and experience of OSPO leadership and team members. This executive support is critical for the culture change that’s required for any successful FOSS initiative.

As previously mentioned, for some companies it may make sense to make the OSPO its own top-level team that reports directly to a chief open source officer (COSO) in the C-suite. This is especially true if the company’s business offerings are highly dependent upon FOSS, such as software vendors or providers of service and support for software packages. The COSO’s position within the organization makes it clear to everyone that FOSS is of critical importance to the business and the company takes this seriously. This

executive placement enables cross-company collaboration and increased visibility. Because this individual must have a great deal of knowledge around FOSS, business strategy, and the company's market and industry, they can provide a strong strategic vision and execute on it.

One thing your company absolutely must not do is to outsource its OSPO and its operations. Certainly it makes sense to bring in outside help when just starting the OSPO team or when facing new challenges, but handing all FOSS operations to an outside organization is a choice that's doomed to fail. OSPO-as-a-Service doesn't work. The effort required to deliver on the OSPO's strategic mission typically requires so much cross-functional collaboration and communication—requiring the building of strong networks within the company—that outsourcing simply isn't feasible. It's also of questionable wisdom to outsource operations that the company has determined are of high strategic importance.

Regardless of whether it forms an OSPO to help manage it, your company must not ignore its software supply chain. In the next part of the book you'll learn more about inbound FOSS and how to make the most of it.

Part II

Inbound (Using) FOSS

The second part of the book covers everything related to free and open source software (FOSS) that you bring into your organization, also known as inbound FOSS.

You'll learn what qualifies as inbound FOSS and how you can use it to the strategic advantage of your organization.

Next, you get operational as you discover how to create effective inbound FOSS policies for your company's needs.

This effectiveness continues as you learn how to avoid inbound FOSS traps related to software supply chain, license and copyright compliance, and FOSS project sustainability.

Recognize and Acknowledge Your Inbound FOSS

In the first part of this book, you learned the building blocks of free and open source software. In this part and the next—outbound FOSS—you will learn how to assemble those blocks in ways that will strengthen and grow your business. Some of these blocks will be tactical—items you can apply for immediate effect—but all of them are important to deliver your company’s FOSS strategy. Depending upon your company’s goals, you will assemble the blocks in ways that can outwit the competition, establish a sustainable path for long-term growth, and provide value to your customers in innovative and cost-efficient ways.

In this part of the book you’ll learn all about inbound FOSS, how to manage it, and how to leverage it in your business operations. Let’s get started.

Inbound Defined

Inbound FOSS is pretty much exactly what it says on the tin: any free and open source software that people are bringing into your organization to use in any sort of business operations. This could be software libraries, databases, or key pieces of software infrastructure that enable your software development teams to build and deploy solutions for your customers or to support other teams. It also could be user applications that are used in day-to-day business operations, such as browsers, email clients, media editing software, operating systems, or any other user-facing tools. If it’s FOSS, it’s used inside your company, and it’s developed by external parties, then it’s inbound FOSS.

You’ll often see inbound FOSS management referred to as “using or consuming FOSS.” In particular, a lot of articles and think pieces out there talk about

“consumption of free and open source software.” These resources usually focus on only the software development side of FOSS usage and tend to overlook the user application aspect.

I prefer the term *inbound* because it’s not yet limited in the same way as “consumption” and therefore is inclusive of all FOSS software brought into your organization for any reason. The term *inbound* also makes a clear distinction between the *use* of externally developed FOSS and the *release of or contribution to* FOSS projects. The latter is outbound FOSS and is the subject of [Part III, Outbound \(Contributing to and Releasing\) FOSS, on page 227](#).

Inbound Isn’t New

The basic concept of inbound relationships isn’t a new one for your company. If you look around, you’ll see that it already has a lot of these that allow it to operate. Some of them may be software related, like Microsoft Office 365 or Atlassian Jira; many of them are not. The company may have vendors that provide cleaning services, audit its bookkeeping, or provide high-speed internet access. Each of these is an inbound relationship with an external entity.

The difference between these inbound relationships and those the company has with FOSS are *intentionality* and *consensuality*. For each of those non-FOSS relationships, your company must take the intentional step of *establishing* that relationship in some form, often by signing contracts for services. Each party in the relationship consents to what those services are, under what conditions the services will be rendered, and how much your company will pay for them. Somewhere inside of your company is a person or department charged with overseeing and maintaining these intentional, consensual, contractual relationships. These things are tracked and managed as a matter of course. To do otherwise, by neglecting or ignoring these relationships, would be business malpractice.

Compare this to inbound FOSS, where intentionality isn’t required to initiate the relationship and consensuality doesn’t exist, because the FOSS community doesn’t get a say in who uses the project or how. FOSS can find its way into your organization in a haphazard way, simply by someone deciding to download and install or use it. No vendor contract is in place, but that doesn’t mean there isn’t a relationship to track and manage. This shifts the burden of intentionality from the front, the initiation of a relationship, to the end—tracking and managing the relationship. It requires a different mindset than most companies are used to, and that mindset starts with recognizing that inbound FOSS exists within their organization at all.

Where to Find Inbound FOSS in Your Company

Your company may have inbound FOSS for many reasons. Being able to identify where FOSS is being used inside of your organization is key to being able to manage it. As you'll see in [Invisible, on page 177](#), sometimes it can be harder than you'd think to locate all of the FOSS in use. Here are a few ideas of where to look for FOSS inside of your company.

Operational Infrastructure and Business Applications

Inbound FOSS can take a critical role in the tooling and infrastructure of your company. For instance, the company telephone tree and branches may run on the Asterisk PBX (private branch exchange) software, providing company conference calling, voicemail, and automatic call routing. Asterisk is Free Software available under the GPL-v2.0 license. Company groupware (email, calendaring, tasks, contacts, notes) may run on the Kopano Groupware solution, a robust Free Software alternative to Microsoft Exchange. Employees may collaborate using Matrix, a free and open source alternative to real-time chat applications like Slack or Google Chat. They may meet with customers using the Jitsi or Big Blue Button FOSS video conferencing solutions.

In addition to operational infrastructure, your company may use various FOSS business-related tools and applications. Employees may use LibreOffice for document creation, spreadsheets, and presentations rather than Google Drive or Microsoft Office 365 applications. They may read their email using the Thunderbird email client or a web-based client by way of the Firefox web browser.

Software Development Dependencies and Infrastructure

Most commonly, FOSS enters your company by way of the software development teams. Development tools such as VS Code and GitHub Desktop are FOSS. So is almost every programming language commonly in use today, such as Python, Go, Rust, and others. Each language typically has an entire ecosystem of free and open source libraries and utilities to support it and provide additional functionality. For instance, if your company's products require complex numerical calculations, it likely uses the NumPy library for scientific computing. It's written using the Python programming language and is open source under the BSD-3-Clause license. Nearly every one of these libraries will themselves rely upon other FOSS libraries, creating large trees of FOSS dependencies. These dependency trees qualify as inbound FOSS for your company.

Software rarely stands alone without its own supporting infrastructure. It runs upon an operating system, and Linux is a common operating system for this purpose. The data that drives the software usually resides in a database like PostgreSQL—available under the open source PostgreSQL license—for access, management, and security purposes. To allow others to access the software, it may be deployed as a container under the open source Open Container specification using the FOSS tool Kubernetes to direct and scale the deployment.

Of course not everything your company creates or delivers is software. Some teams, such as marketing, support, and community, may release content to help your customers accomplish what they need. Creating that content is a lot easier thanks to assets like documents, images, music, and videos shared openly under Creative Commons licenses.

Free and Open Non-Software

If FOSS is free and open source *software*, why do I include non-software things like documents, images, music, and videos in your company's inbound FOSS? Colloquially, many people lump works shared under Creative Commons licenses in with FOSS. This convenient shortcut can lose sight of Creative Commons' effort to promote the open and transparent sharing of creative non-software works.^a

It's true that assets released and shared under Creative Commons (CC) licenses aren't software, but the ethos of CC is close kin to that of FOSS. Both movements share the goal of allowing people, companies, and organizations to share their work in open and standard ways that enable and encourage reuse. But unlike FOSS—which has no single central authority—the Creative Commons nonprofit is the single central steward and shepherd of the CC movement.

In the case of inbound, CC assets require similar treatment to that of FOSS components. Both require attribution to the original creator(s), have license variations that require the attention and compliance of your company, and therefore must be tracked by the company in some way. So while initially it may not look like you should include CC works in the software-shaped FOSS bucket, it's logical and smart to include both CC and FOSS in your inbound policies and procedures.

a. <https://creativecommons.org>

When considering the matter of inbound FOSS in your company it's easy to get mired in the software development space, but as you'll see, FOSS can become critical in all areas of the organization. Casting a wider net than software development can allow you to find and make the most of all of the inbound FOSS rather than a fraction of it.

Inbound Stakeholders

As you read through these chapters on inbound FOSS and consider applying what you learn, keep in mind that your actions won't occur in a vacuum. Inbound FOSS already exists in your corporate ecosystem, meaning there are people who rely upon it to get their jobs done. Making changes without consideration for how it will impact people's workflow is at best going to irritate them and at worst going to lead to disaster. As with all matters touching upon free and open source software, you'll get the best results if you enter into it with a mindset of collaboration from the very start. When you think about creating and implementing new policies, consult those whom it'll affect. Gaining buy-in and support from these stakeholders will ensure everything will go more smoothly.

Legal Team

Due to the obligations enshrined in every FOSS license, include your legal team in all inbound FOSS discussions. Getting them involved early will prevent time-consuming mistakes from overlooked or unknown legal constraints. Likewise, if your company operates in a highly regulated industry like financial services, consult with members of your compliance and audit teams before implementing any policies.

You may be reluctant to involve legal and compliance in the inbound FOSS process for fear that they'll slow and obstruct the process, but if they're involved early on, then they'll have the opportunity to provide the input that will allow everything to proceed smoothly. What others see as obstruction is simply these teams doing their jobs, ensuring the company doesn't end up in legal hot water. If they're included from the start, then they can do that job at the same pace as everyone else involved with the inbound FOSS process rather than having to hold everything else up at the end because they weren't given the chance to have input.

Security and IT

Your security and IT teams are involved when acquiring a new vendor for proprietary software (or should be), and inbound free and open source software is no different. These teams are responsible for rolling out and maintaining technology across the organization and for ensuring that any known security vulnerabilities are managed appropriately. They can't do their jobs if they're not aware of what software the company is using. Involving them in inbound FOSS policy planning gives them the opportunity to guide the company toward

more maintainable and sustainable options while also giving them the information they need to evolve their own processes in an efficient manner.

Software Developers

Where inbound FOSS is concerned, the software developers usually are the ones most impacted by any changes in policies and procedures. The chances that they're using FOSS in their work are pretty much 100 percent, so any change to policy is going to have an outsized impact on them getting their work done.

Furthermore, these people are very familiar with software and technology. In my experience, when they're faced with a policy that prevents them from doing their jobs (or sometimes simply one they don't like), they *will* find a way around it. Developing inbound FOSS policies and procedures without input and buy-in from the software development team is a good way to waste everyone's time and lose credibility.

As well, any policy or procedure needs to take into account the inbound FOSS that already exists in the software development process and workflow. If it doesn't, then you're likely to end up in a situation where the policy restricts use of some FOSS that's already present in and critical to the business. Rearchitecting software to remove or replace critical pieces of FOSS is an expensive proposition and not something to enter into lightly. Including representatives from the software development and delivery teams in discussions about inbound FOSS will ensure that everyone's needs are met without risk of disrupting workflows.

Marketing, Product Management, Tech Writers, Designers, and Others

Many teams are frequently forgotten when discussing inbound FOSS, since at first glance it doesn't appear as though they're stakeholders. Just as frequently, that assumption is wrong. Not only are others in the company impacted by inbound FOSS, but they can also *have* impact, ensuring inbound initiatives are more successful.

The marketing team can explore inbound FOSS usage to tell brand-strengthening stories. Product management can draw insights from the communities of the inbound FOSS projects to influence product-market fit. Designers often use open source frameworks and tools to create their works. Tech writers are critical for ensuring that all policies and procedures are accurate as well as easy to understand and locate.

In summary and to be completely frank, no one likes being left out of conversations that may impact them, and they're likely to resent it if they are. Don't assume you know who the inbound FOSS stakeholders are; confirm it. Look around your company to see which teams might have some stake in how inbound FOSS is used inside the organization, then give them a seat at the table. Include them in conversations from the very start, listen to their feedback and incorporate it as makes sense, and you'll minimize people digging in their heels and resisting the policies later when they're launched.

Software Supply Chain

Bees4Less, your one-stop pollination station, is very aware of its particular supply chain. Its leaders pay attention to whether there are issues with the farms where they store their bees. They know whether there's flooding this season at the gardens where they source their flower seedlings. They keep enough hive-building materials on hand in case someone wedges another container ship in the Suez Canal and delays shipment of the parts they need. All of these things are links of the *supply chain* for Bees4Less. If any link in that chain should break, Bees4Less could get stung.

Your company also has supply chains. These naturally will vary depending upon the company's market and business strategies, but inevitably one of those chains is the *software supply chain*. This chain is made up of many links, and each link is a different piece of software that's critical for the functioning of the business and for delivering value to your customers. Your inbound FOSS forms links in this chain and, if your company is like nearly every other one, these links are invisible. What happens if one of those invisible links should break?

Until recently, few companies had much insight into the inbound FOSS links in their software supply chain. As free and open source software gains more mindshare in the corporate world, all that is changing. Not only are companies starting to become aware of the importance of these links in the chain, evildoers are as well and are taking advantage of unpatched security vulnerabilities in FOSS components to compromise private computer systems. Staying on top of security updates is the only way to avoid this risk, but it's impossible to do that if you can't see all the links in your software supply chain.

Security vulnerabilities aren't the only concern where software supply chains are concerned; software sustainability matters as well. FOSS or proprietary, when acquiring software your company should always consider whether the source of that software is managed in such a way that it'll exist for as long

as your company needs the software. If that source isn't stable, then relying on the software is a risky proposition.

Awareness of and attention to your company's software supply chain allows it to make informed decisions about which software to use. In the case of inbound FOSS, your company also has the opportunity to take action and ensure that the communities that develop these critical links in their software supply chain are strong and healthy, ensuring the software will be around for years to come.

The reality of your software supply chain—FOSS or otherwise—is complicated. You'll learn a lot more about how to discover and maintain that supply chain in [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#), and [Chapter 13, Sustain. Maintain. Retain., on page 213](#).

Now that you've learned what inbound FOSS is, let's talk a bit more about how it can become a vital element in your company's strategy.

Inbound FOSS in Your Corporate Strategy

For better or worse, there's a lot of room for improvement where corporate FOSS strategy is concerned—any corporate strategy, really, but FOSS strategy in particular. In my experience, most corporate approaches to free and open source software is, at best, purely tactical and reactionary. FOSS rarely plays a decisive role in overall corporate strategy, which is unfortunate, considering the oversized role it plays in the success of business operations for most organizations. Typically, if a company has acknowledged FOSS as a driving force in the business at all, it's still relegated to an understaffed and underfunded corner of the org chart, fighting the good fight to help the company do the right thing for itself and for the communities on which it relies. Even in these cases, the approach is largely tactical.

In the few cases where I've seen companies form real strategies around free and open source software in their businesses, those strategies usually focus on *outbound* FOSS—the FOSS the company creates and releases for others, or the company's contributions to externally created FOSS. As you'll learn in [Chapter 16, Strategic Benefits and Risks of Outbound FOSS, on page 243](#), outbound FOSS is an important part of any business strategy, but it's only half of the picture. Inbound FOSS also has an important place in corporate strategy and, as you'll see in this and the following chapters, its strategic role is only growing.

Role of Inbound FOSS in Digital Transformation

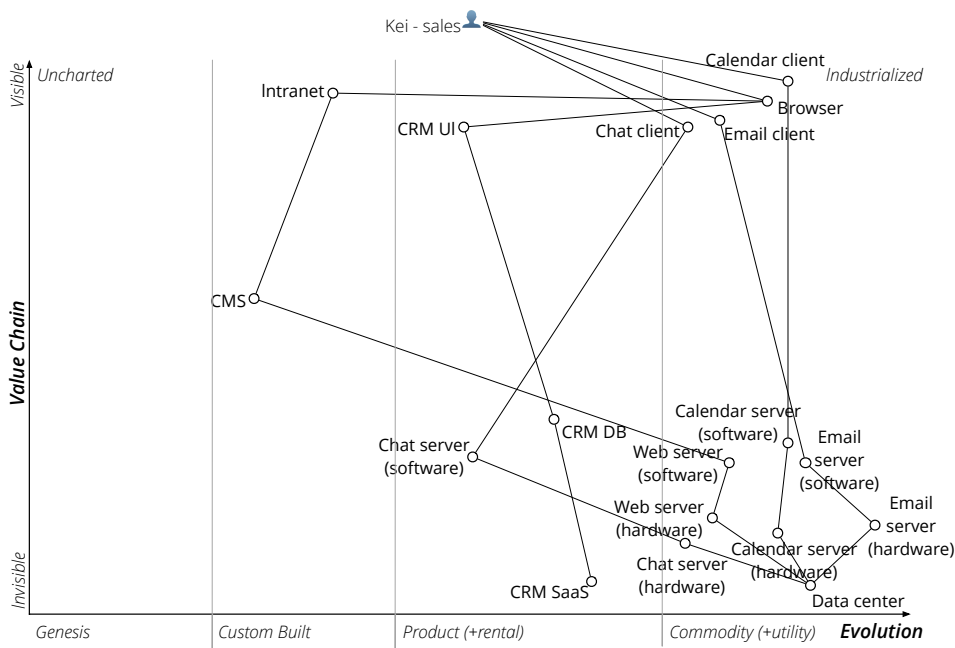
One of the largest movements in business today is toward *digital transformation*. This is a customer-focused and technology-enabled evolution of the company such that it becomes digital-first in its operations, business models, and all other functions. Companies engaging in a digital transformation effort aim to automate processes, increase efficiency, decrease time to market (TTM), become more nimble in their response times, and uncover innovation opportunities.

Digital transformation takes a holistic look at what's going on in the company and its business, so it therefore touches upon all aspects of both. Successful digital transformation sometimes requires dramatic culture change. For that and many other reasons, it's hard to do right (or at all), but the companies that make it work reap immense benefits that make all that effort worthwhile.

While the entirety of the complex topic of digital transformation is outside the scope of this book, the role inbound FOSS can play in your company's digital transformation effort most certainly isn't. When viewed through a digital transformation lens, developing a managed approach to inbound FOSS—rather than the accidental or incidental one so common in companies today—creates a strategic pillar for those efforts. Intentional action to select and maintain FOSS components when transitioning from analog to digital, or when modernizing legacy approaches, can establish a strong foundation that gives your company the freedom, modularity, and flexibility to react to whatever market conditions throw at you. Let's look at an example.

Inbound FOSS and the Digital Transformation of Bees4Less

Kei, a salesperson at Bees4Less: The Pollination Station, uses many different tools for their job. Here's a Wardley Map¹ representing a slice of those tools:



1. <https://learnwardleymapping.com>

You can see that the usual suspects are all there in the lineup:

- Kei naturally uses their email and chat clients (desktop and mobile software) a lot, along with their calendar. No bee salesperson can do their job without these anymore, can they?
- Kei also uses the company's intranet for storing and retrieving white papers, product summaries, sales decks, and case studies.
- The intranet runs and relies on a purpose-built content management system (CMS).
- Kei tracks all of their pollinator sales leads in a customer relationship management (CRM) tool. It's a software-as-a-service (SaaS) subscription but has been highly customized for Bees4Less's sales workflow.
- Each piece of this software relies on its own server software and hardware, and many include their own database (DB) as well. (For the sake of simplicity on what's already a complex diagram, the DB hasn't been split into its software and hardware server components.)

Items that are more visible to Kei are closer to the top, and items become less visible to Kei as you go down to the bottom of the map. Things that are more custom or less evolved are to the left, while more “off-the-shelf” or common-place things are to the right of the map (see also the [sidebar on page 134](#)).

As a part of Bees4Less's digital transformation effort, it's reviewing its existing sales-related software systems, looking for elements that can be upgraded and for ways to make the entire process easier and more efficient for Kei and their customers alike. Even in this small slice, you can see a lot of moving parts to consider.

When speaking with Kei and their colleagues, the company learns that any changes that will require them to use different email or calendar clients will lead to not only needing more time to close sales (as the team learns how to use the new software) but also to expressing a lot of anger. The sales team have strong opinions about their software tools and have optimized their workflows around them, so any transformation effort should change these pieces last—if at all. While Kei is attached to their email and calendar clients, they have no strong feelings about chat. They mostly just wish everything would work together smoothly and that it all integrated with the CRM.

The various pieces of server software, aside from the SaaS CRM, are all hosted in a data center on server hardware. Each piece of software is maintained by the internal IT team, who has outsourced the care and feeding of

What's That Weird Diagram?

"I have a map of the United States...actual size. It says, 'Scale: 1 mile = 1 mile.' I spent last summer folding it. I hardly ever unroll it. People ask me where I live, and I say, 'E6.'" —Steven Wright

Those funny little diagrams with the dots and lines are Wardley Maps. Invented by researcher Simon Wardley and released under a Creative Commons license, Wardley Maps are a tool for modeling value chains and are useful when discussing and formulating strategies. It's a picture representing relationships and a tool for framing discussions. Wardley Maps can help people better picture the entire landscape and identify possibilities that might otherwise remain hidden.

Wardley Maps are *tools*. Like all tools, they don't have agency and can be misunderstood or used incorrectly. To quote George Box, "All models are wrong, but some are useful." They don't create answers, only facilitate their discovery. That's how I'll use them in this book: to frame and facilitate discussions when its helpful for visualizing examples.

One of the advantages of Wardley Maps is you don't need to understand the mapping method to get value out of the map itself. All you need to do is look at the picture and talk about it. What can you see? Are there relationships? Could any of those dots be moving in a specific direction? Are any missing? How would what you see impact your business? These questions are just the start of the conversations that a Wardley Map can facilitate, even without studying the mapping method itself.

You can learn all about Wardley Maps at the aptly named site, *Learn Wardley Mapping*.^a

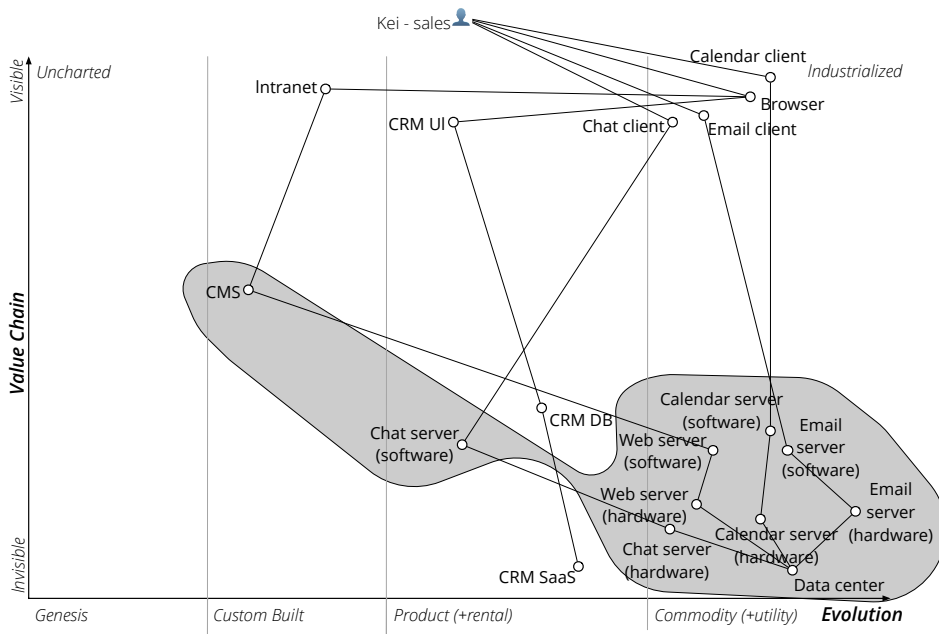
a. <https://learnwardleymapping.com>

the hardware to staff at the data center. Bees4Less pays annual license fees for the proprietary email, calendar, and chat server software along with its yearly service contract for the data center and for the SaaS CRM.

The intranet CMS—being purpose-built in-house—doesn't have license fees, but it comes with hidden but considerable maintenance costs. The people who developed the CMS moved on from the company years ago and no one else really knows how it works. Anything beyond a simple content update requires weeks of work and testing, so no one wants to touch it at all. This is a big pain point for just about everyone at Bees4Less, and they all want it fixed, as shown in the [map on page 135](#).

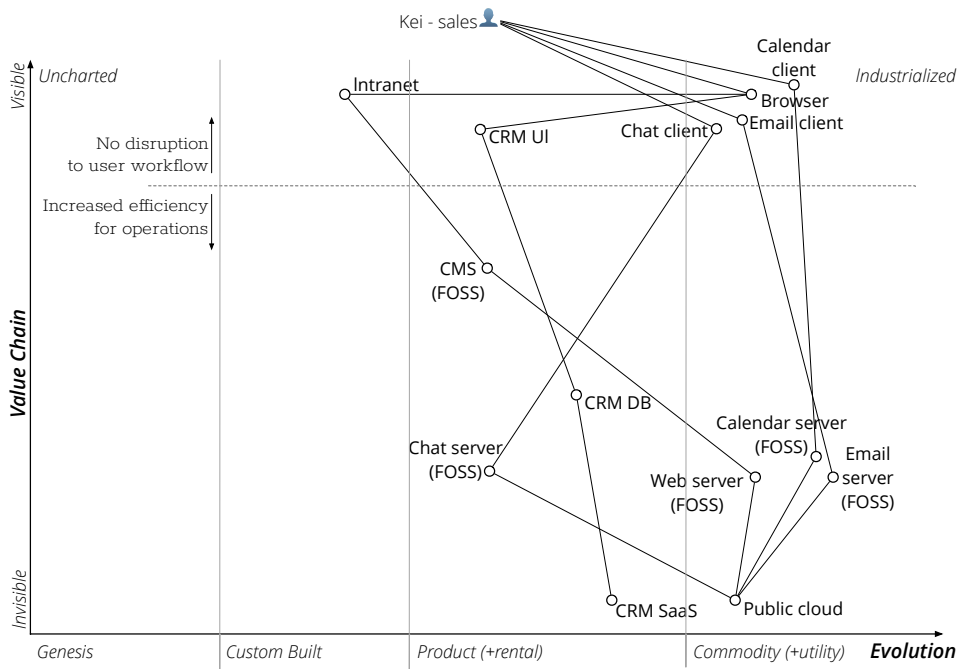
After studying the overview of their infrastructure, Bees4Less decides to make some changes to support their digital transformation:

- They choose to get their servers out of the data center and into the cloud.



- They considered doing a “lift-n-shift,” transitioning to cloud-based versions of their existing server software but mostly opted to migrate to new, modern server software to allow them to stay up-to-date with technological advancements rather than being held back by their clunky legacy tooling. This modernization will allow the company to remain technologically nimble for years to come.
- To increase their future options, Bees4Less also makes it a requirement that each modernized piece of server software must be open source. The servers themselves may be SaaS hosted and maintained by external vendors, but requiring the software to be open source ensures the company can move between vendors if needed, or even take over hosting the servers themselves.
- Another requirement Bees4Less sets for the modernized server software is that each server must use open standards and provide open application programming interfaces (APIs). The open standards will allow Kei and their colleagues to continue using the client software on which they already rely, while the open APIs will enable integration between all the pieces, improving workflow and visibility in the overall system.
- The APIs and integration also open the door to automating some of that workflow, providing even greater efficiencies and freeing up more of Kei’s time to focus on the most important thing: their customers.

After these changes, the Bees4Less infrastructure map now looks like this:

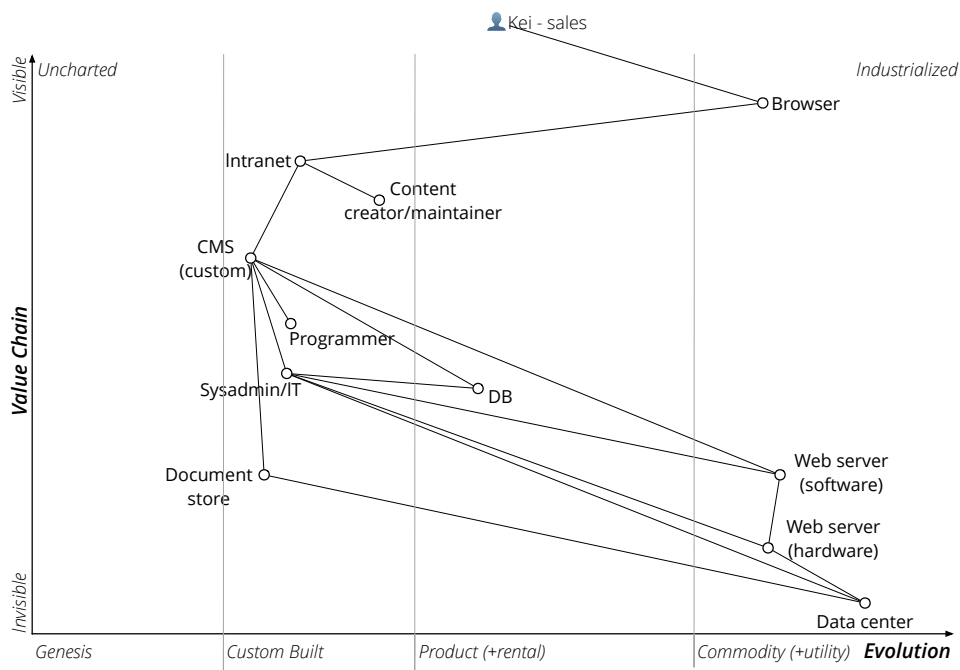


This involves tradeoffs, of course. Bees4Less trades high-touch control over its software for subscription fees on commodity, cloud-based products, among other things. Does this tradeoff make sense? For some companies, yes, it may, by bringing workflow improvements, increased productivity and customer satisfaction, and perhaps lower costs. For others it may not be the right move. For strategic or regulatory reasons, they may need to retain their data center and the software in it. The important part of this example is not the conclusion but the process Bees4Less went through to get there. Before making the decision, they looked at the big picture. This level of intentionality allowed them to see possibilities they may not have discovered otherwise.

Lower Expenditures Through Intentional Use of FOSS Alternatives at Bees4Less

Now let's look at a single piece of the preceding example in more detail: the intranet. Here's a [map on page 137](#) of the most conspicuous parts of Bees4Less's original intranet.

Remember, Bees4Less has an intranet running on an internally developed, purpose-built custom CMS. In software, we say that they "rolled their own" where their content management system is concerned. Because it's custom

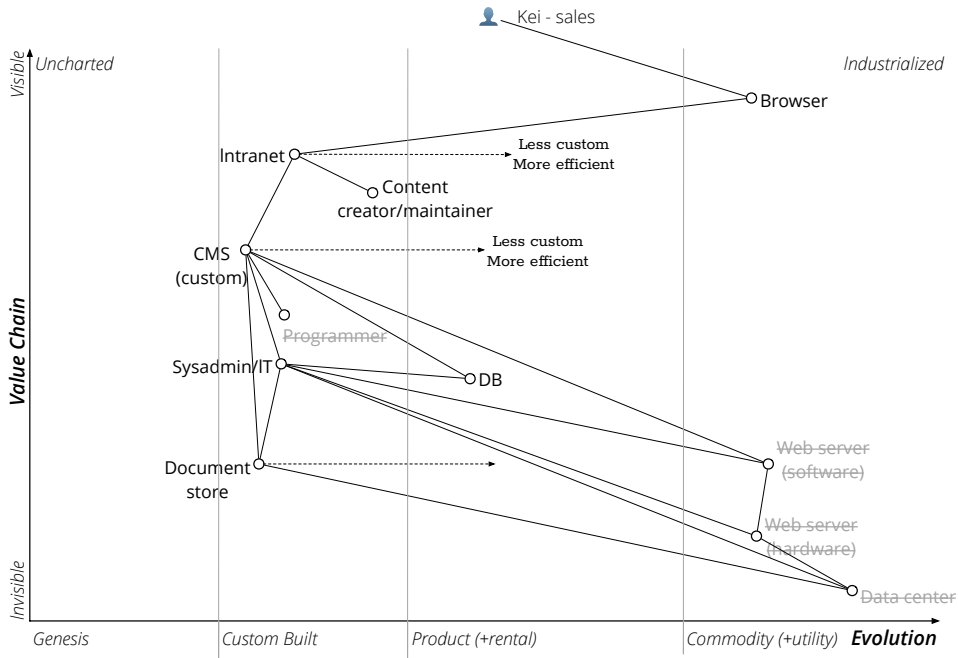


software, the only people who knew how to use or maintain it were inside the company, but most of them have left over time. This makes it difficult to onboard new staff members to update the intranet content or maintain the CMS software. The software itself is based on technology that made sense at the time the project was created but has since fallen far behind the times, making it difficult to add new features or integrations. Additionally, when new features *are* added, they need to be metaphorically bolted onto the side of the main software.

The end result is like Frankenstein's monster trapped in a crumbling castle: no one wants to get too close to it, but they have to shore up the walls to keep the monster from wrecking everything. On top of all that, the software runs on dedicated hardware in a data center, requiring staff to manage the hardware and ensure it remains functioning and healthy.

Looking at the [map on page 138](#) and reviewing the preceding description, several items jump out:

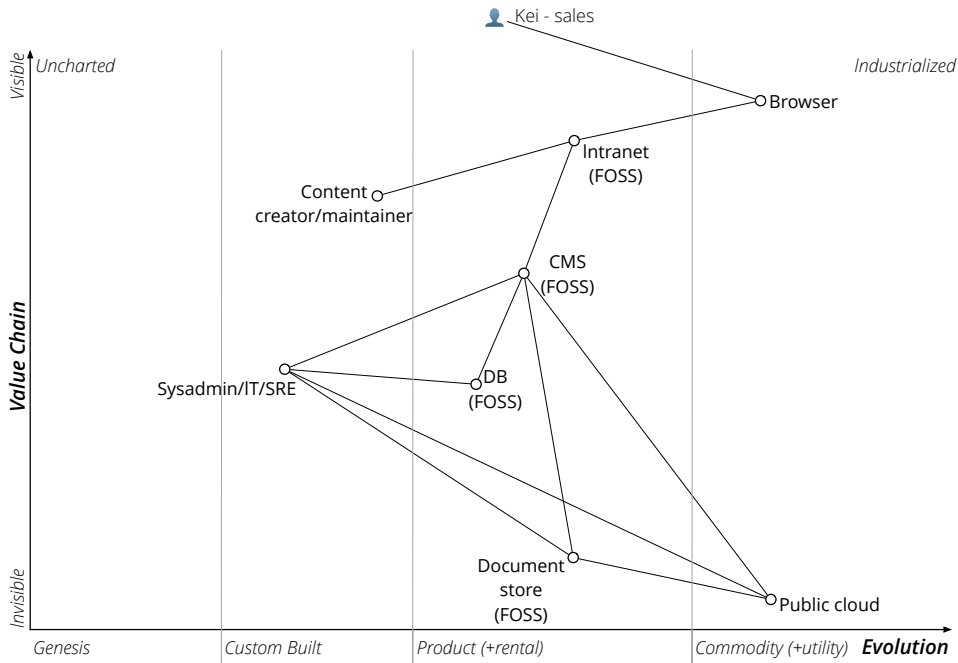
- Increased cost for maintenance of the CMS software.
- Decreased productivity due to the CMS software learning curve, fragility, and lack of integration.
- Capital expenditure for server hardware.



The first two of these are considerable problems, but they both are solvable by Bees4Less transitioning to use one of the many robust FOSS CMS software packages that exist. Moving the content to something like WordPress, Drupal, or Plone not only removes the expense of needing to build and maintain a home-rolled CMS, it also decreases the amount of time needed for individuals to onboard. These FOSS CMS solutions have been around for decades now, so it's likely that staff that join the company will already be familiar with the chosen solution. Even if they're not, venerable and reliable FOSS software packages like these have equally venerable and reliable documentation; it's unlikely that the original home-rolled solution could say the same. These FOSS CMS packages will provide access controls, APIs, and a large ecosystem of community-contributed plugins for every type of integration under the sun. Because of their broad use, each CMS also includes functionality for internationalization and accessibility of content, which is another thing the home-rolled solution is unlikely to provide.

As for the server hardware and data center, they're no longer needed. It's now 20 years since Amazon launched what became Amazon Web Services, kick-starting the cloud computing movement. Cloud computing is trustworthy, reliable, and now the standard mechanism for hosting applications of all sorts. Bees4Less could choose to create their own private cloud using the

FOSS OpenStack project, but this would be overkill for something like their intranet CMS unless they have some very special constraints and requirements. It's much more likely that Bees4Less can run the CMS on services provided by one of the many public cloud vendors available today.



With this approach, Bees4Less has accomplished the following:

- They've eliminated the need to maintain a fragile piece of software.
- They've improved onboarding times and staff productivity.
- They've converted the capital expenditure (capex) of server hardware to the operating expenditure (opex) of on-demand cloud computing.

Sounds pretty good, right? Right! Except, as always, there are trade-offs.

Recall that back in Chapter 4, [Maintenance Costs, on page 83](#), you learned that free and open source software is free as in puppy. You may not need to buy it or build it, but you *do* need to take care of it. This means Bees4Less will either need to hire people who know how to administer their new FOSS CMS and cloud computing services or they'll need to provide training to their existing staff so they can do this work.

Also, we can't forget that this transition won't be the work of a weekend. Doing it properly, minimizing the impact on Kei and especially on their customers, will take planning and time. It also may require the experience of someone

like an information architect to clean up that CMS content and make it easier and more accessible to use. Like all digital transformation efforts, this one may take years for Bees4Less to complete, but in the end the company will find itself in a better position to flex and grow in the future.

Avoid NIH Syndrome

The previous scenario isn't only a good example of how moving to FOSS solutions can help lower expenditures, it's also an object lesson in avoiding NIH syndrome. This is a common malady in IT and software departments where the team is convinced that they must build everything from scratch because they can't trust anything that's Not Invented Here (NIH).

NIH syndrome may not kill a company, but it certainly will disable it. The resources and budget required to build tooling from the ground up are considerable, but the cost of maintaining that tooling over the years is exorbitant. These are resources that could be moving the company forward on strategic initiatives but instead are dedicated to what rarely ends up being a good investment over time.

NIH syndrome can sneak in anywhere in a company, but as mentioned previously is most prevalent among the technical staff members. Programmers like to build things, and often they're convinced that they can build those things better than anyone else. While that may be true in some cases, these statements don't usually consider the costs or strategic value of that building process. Unfortunately, where technological solutions are concerned, these well-meaning builders' voices can hold outsized influence in a build-versus-buy decision for software.

The growth and maturity of FOSS gives companies a third option in that build-versus-buy discussion: install and modify. By establishing guidelines for software procurement such that FOSS solutions are guaranteed fair consideration (you'll learn more about this in [Chapter 10, Get Started with Your Inbound Policies, on page 163](#)), companies can avoid the tragically expensive mistake of building all their own tooling. Instead, by insisting upon review of FOSS alternatives, companies find that a FOSS package either meets all of their requirements or is a close enough match that they can choose it. Then, because the nature of free and open source software ensures the freedom to modify it, they can customize it to get them the rest of the way to their goal.

The build-versus-buy question is thus resolved. If the company also then submits those modifications back to the project (see [Chapter 17, Master Contribution Fundamentals, on page 263](#)), the maintenance costs drop to nearly nothing, as they're shared across all users of the software. The company also avoids an unmaintainable mess by trying to keep their internal modifications in sync with the upstream FOSS project as it receives updates. By not using all that development effort for the creation and upkeep of custom-built software, the company can then have their software development team work on projects that are more critical to its business.

In some situations, the answer to the build-versus-buy-versus-install/modify question is “build...and then release.” In [Chapter 19, The Strategy of Releasing FOSS, on page 297](#), you'll learn how FOSS is a powerful strategic lever that can move or even create new markets with that custom-built tool.

More Rapid Innovation and Time to Market

NIH syndrome affects much more than internal tooling; it also affects development of products. Programmers love to build things, even if those aren't necessarily things that needed building in the first place. Some programmers also are blessed with an abundance of self-confidence—bordering on bravado—and are convinced that they're capable of creating better software than what one could acquire externally. They may be right about this, but they're rarely correct that they can bring about that act of creation on a timescale that allows them to deliver value to the customer by the planned launch date.

Some programmers also feel that any development process that doesn't have them building everything from the ground up must be cheating or taking shortcuts. These completionists may deliver a coherent and tightly integrated system, but it will take them years to do it. The fact is, no matter how good you are at what you do, you can't do it all. Thankfully, for software, FOSS is here to help.

Some companies, when faced with a need for new software, have policies that require teams to look first at FOSS options before even having the discussion of whether to build or buy. Having these FOSS-first guidelines for software procurement not only helps your team avoid being struck by NIH syndrome, it also helps your company serve your customers better and more quickly. By assembling preexisting FOSS components, you can get new functionality

in front of your customers in weeks rather than years, providing value and reducing churn. Why would they leave and go to your competitor when your company is rapidly delivering what they need?

According to a 2022 report by Synopsys, 78 percent of the average proprietary software package is actually FOSS code.² That's 78 percent of the software that your team doesn't have to write. Using these FOSS components as the fundamental building blocks of your product software gives your product development process a huge boost, giving your company's strategic ideas shape quickly while dramatically reducing the time it takes to get your product to market (TTM).

Reduce Cost of Maintenance

When an artist creates a statue, she then displays it somewhere and admires the finished product. When a programmer creates software, she then has to maintain it. There's no such thing as a "finished" piece of modern software. You'll have bugs to fix, features to add, security patches to apply, and other necessities that ensure a piece of software is anything but a static creation.

If your company, stricken with NIH syndrome, has created all of its software from scratch, it then retains 100 percent of the responsibility for maintaining it for as long as it remains in use. This maintenance can be expensive, and it can consume programmer time that could otherwise go toward developing new features and products for your customers. But if your company has inoculated itself against the syndrome, it will have used FOSS components to build its software. These components are a common and shared resource for the entire ecosystem, meaning the maintenance burden is common and shared as well.

All users of a FOSS component have the responsibility to contribute bug fixes and features back to the community. This ensures the stability and longevity of the FOSS project, but it also distributes the maintenance costs and burden across many individuals and institutions. Everyone benefits equally when everyone contributes equally, but even when some shirk their responsibility and freeload off the work of others, the overall community benefit is still considerable.

Imagine your company as one of those with 80 percent of its software composed of FOSS. That's 80 percent of its software with a shared maintenance burden. This frees up a meaningful amount of programmer time for strategic initiatives, helping your business evolve and succeed more efficiently.

2. <https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html?intcmp=sig-blog-ossra22>

More Secure Solutions

As you learned in the last section, there's no such thing as a finished piece of modern software. Similarly, there's also no such thing as a bug-free piece of modern software, and some of those bugs are security vulnerabilities. This holds true for all types of software, proprietary or FOSS.

The perception of FOSS as inherently insecure is a grizzled and tired myth. For decades people faced with using software created under the open development model have resisted with largely unsubstantiated claims about poor security. We can trace some of this reaction back to NIH syndrome ("if it wasn't built here, it's no good"), but a lot of it goes back to the age-old human fear of the new and unfamiliar—software xenophobia, if you will. FOSS isn't like the proprietary software they know, so therefore it must be dangerous. As is often the case with xenophobia or other forms of bigotry, looking at the data reveals the reaction as fit for the trash heap.

Over the years many studies have shown³ that FOSS tends to have fewer vulnerabilities than proprietary software, and when those vulnerabilities are discovered, they're fixed with remarkable speed. Much proprietary software has the advantage of dedicated teams of quality assurance (QA) and security professionals, which should not be discounted. These people do good work, but they're doing it behind closed doors. In contrast, free and open source software projects are developed entirely in the open. All of the code and conversations are public and therefore open to audit and inspection by anyone. Contributions to the code base receive peer review before they're merged into the software.

With their work displayed for all to see, developers take fewer shortcuts in many FOSS projects. The overall openness of the free and open source software world enables a level of scrutiny that's impossible for proprietary alternatives and allows for the discovery of more bugs before they have the chance to become problems. When those problems do arise, the community-driven development model leads to rapid fixes of critical vulnerabilities for the good of all who use the software.

Thanks to these advantages, with appropriate maintenance of your company's FOSS supply chain, it can use free and open source components with confidence their software won't undermine their strategic initiatives or compromise their customers' safety. You'll learn more about FOSS supply chain management in [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#).

3. <https://www.redhat.com/en/enterprise-open-source-report/2022> and <https://www.govtech.com/security/is-open-source-software-more-secure.html>

These are a few examples of how you can use FOSS for strategic benefit within your company. The options—and the opportunities—are limitless, but they don't come for free. As with all strategies and business transformations, it's not a magic pixie dust that you can sprinkle on things to make them better. It takes a lot of thought and effort to make inbound FOSS work for your organization. In the following chapters you'll learn about how to do this, including how to sustain your FOSS supply chain ([Chapter 13, Sustain. Maintain. Retain., on page 213](#)), and how to stay in compliance with FOSS licenses ([Chapter 12, Maintain FOSS License Compliance, on page 201](#)). All of these changes stand on a foundation of policies and procedures. You'll learn about that next.

Establish Effective Inbound Policies and Procedures

Say “policies and procedures” to many people and their eyes start to unfocus, their mind wandering to the land of Anywhere but Here. Through our treatment of policies and procedures, we as humans have given them a bad reputation to the point where many people mentally check out when faced with them. We make them difficult to understand, difficult to follow, and difficult to care about. This is a challenge for companies looking to embrace inbound FOSS within their organization, but there’s hope.

The two most common mistakes I see with companies’ inbound FOSS policies and procedures (P&P) are either making them too restrictive and cumbersome, which encourages people to ignore or work around the P&P to get their work done, or neglecting them altogether to create an inbound software free-for-all. Both of these mistakes have the same result: potentially risky software sneaks in, but no one knows about it. In this chapter, you’ll learn how effective P&P can minimize risky software sneaking into your organization.

Why We Need Policies and Procedures

Policies and procedures surround us, whether we acknowledge them or not. We need to get stuff done and, being the clever apes we are, will come up with a method to do that (even if we make up the method as we go along). This method is a *procedure*: when the light bulb burns out, I turn off the light switch, twist the bulb to the left, remove it, put a new one in its place, twist to the right, and turn the switch back on—and I’m rewarded with light. We also develop a rubric or protocol for when it’s appropriate to apply that procedure. This

rubric is a *policy*: when the light bulb burns out, I don't replace it if the power switch is still on, especially if I'm standing in a puddle of water.

As you see in the preceding example, policies and procedures (P&P) don't need to be formal or even acknowledged; most of the time they simply exist. When you change your average light bulb, you don't grab a manual and read up on how it's done—you just do it. The procedure exists, but it's not written down. You learned it at some point and now you practice it regularly. P&P also evolve as you learn new things. It's not much of a stretch of the imagination to consider why the policy for changing a light bulb includes that bit about not standing in a puddle of water, to picture a time when it wasn't part of the policy, or to puzzle out what may have happened for it to be added.

Changing a light bulb is a basic task and doesn't usually need to be formalized, but many of the tasks we perform are much more complex. That complexity compounds when other people are involved in some way. The outcome of that complex task impacts others, whether they're involved in performing the task or not. It may impact the business. It may impact your customers. Suddenly, it becomes important not only that someone performs the task but that they know why, how, and that the outcome is predictable. Now you need a documented policy and procedure for that task: it's complex or important enough that its successful outcome must be ensured.

Policies and Procedures Are Business Assets

Policies and procedures are the “how we do things here” of your company. This is vital institutional knowledge that allows business as usual to be, well, usual. P&P make the implicit explicit; they say the quiet part out loud. Properly handled, P&P get everyone on the same page and help you set expectations to which you can then hold people. Policies and procedures are a safety net that allow new employees to do the right thing and existing employees to do new things. Companies with unhealthy relationships with their P&P can't scale, have poor onboarding experiences, and will be in crisis mode if Isabel from payroll wins the lottery and moves to the Maldives without training someone else to do her tasks.

Many companies either neglect their P&P altogether—often under the guise of being “flat, nimble, and low-overhead”—or micromanage them to death, both in the documenting and the enforcing stages of the P&P life cycle. Both of these approaches are unfortunate and unhealthy, but both also are fixable.

Pretending a policy or procedure doesn't exist does little to negate that existence, but it does increase friction within an organization. If your company

has assigned tasks to team members but hasn't provided any guidance for how or why these tasks should be performed, you can rely on them coming up with their own way to get their work done. This can be valuable: every procedure has room for improvement and evolution. But it also can be problematic: coordination and cooperation are difficult when people are approaching a problem in different ways. This can lead to frustration and perhaps even to failure.

That frustration reaches beyond the line workers into management. When they assign a task, often managers have their own idea of how it should be performed and what the outcome should be. If that idea doesn't match those of the people performing the task, then the manager may get angry at the team and scold them for poor performance, despite not expressing the expectations to which they're holding their staff.

Bad P&P Are Worse Than No P&P

Bad policies and procedures may be one of those situations that defy the "better than nothing" aphorism. We have poorly written and enforced P&P to thank for peoples' negative perception of *all* policies and procedures. Jargon-laden verbiage, out-of-date or flat-out incorrect content, and unsubstantiated or nonexistent reasoning are just a few of the all-too-common characteristics of corporate policies and procedures. The sad state of the contents of much corporate P&P makes enforcing them consistently difficult, if not impossible. It's no wonder most employees see P&P as a burden rather than the helpful asset that they should be.

Companies that view their P&P as the assets that they are take the time to create useable, reliable P&P documents. These companies trust and empower their employees to do the right thing even in unforeseen situations. This, in turn, supports efficiency across all aspects of business operations. The trust it affords the employees can lead to lower staff turnover. Overall, developing a healthy relationship with policies and procedures enables companies to do business better.

Make Policies Useful (and Used)

A healthy relationship with policies and procedures starts with documentation. Nothing scales as well, is as usable and accessible, and is as vital to efficient business operations as well-written and well-maintained P&P documentation. On the other hand, if these are handled poorly, then nothing is as frustrating. In this section you'll learn some tips for creating and maintaining this important business asset.

Consult the People Who Will Use the P&P

All too often, policy and procedure documents are written by people with good intentions but who are too far removed from what they're documenting to understand it. When creating or updating P&P docs, always speak with the people who are doing the work.

Even if your company has no existing P&P documents, people have already created policies and procedures to get their work done. Talk to them. Listen to them and watch how they perform tasks, learn the criteria for when certain steps should happen and why, and understand how the task is done before writing the documentation.

Once that P&P documentation exists, have these people give it a final review to make sure the writer understood the P&P correctly. Then, if you really want to ensure the P&P are useful, hand this documentation to someone who's never done the job to see whether any steps are missing or unclear in the P&P. This ensures that the P&P documentation reflects reality rather than being, at best, an informed guess or riddled with assumptions.

Standardize Your P&P Format

As the company assembles its P&P documentation, it can help to use a standard framework or format for those documents. A standard format makes it easier for people to locate what they need in a document, and having a starting place like this avoids the mental paralysis that comes with a blank page. Looking ahead, having a standard format for the documents will make the process much easier when, inevitably, the documentation must migrate to a new platform or framework.

The format doesn't need to be anything formal and should include only the information people need to use and understand the policy or procedure. For instance, a policy format could include these sections:

- Policy/procedure name
- Any pre-reqs for using the policy/procedure
- The contents of the policy/procedure itself
- Reasoning behind the policy/procedure
- Contact information if there are questions about the policy/procedure
- Date (year, month) of policy/procedure publication or creation

Regardless of the format selected, always make sure it includes some sort of reasoning behind the policy. It's easy to overlook including this information,

and some companies will even actively disallow its inclusion rather than trust their employees with the information. That's a shame, as providing the reasoning teaches people who use any part of the P&P why it exists and what it's intended to accomplish, empowering them to use their judgment when they come across edge cases and exceptions. By including the reasoning within the policy or procedure itself, people can respect the spirit of the policy or procedure even in situations when they're not able to follow it to the letter, or they can identify P&P that you should remove because their reasoning no longer holds true.

Here's a brief example drawn from the Bees4Less P&P collection:

Processing Return Requests for Bees

If a customer is unsatisfied with their bees, follow these steps to issue a refund.

Requirements for this procedure:

- Be authorized to use the admin functions of Hive (the fulfillment system).
- Know how to use Hive ([click here for instructions](#)).
- Know how to use Hon-E (customer ticketing system; [click here for instructions](#)).

Steps to issue a refund:

1. Use their account name or order number to find the order in Hive.
2. In the Staff Comments box, add any reason the customer provided for requesting a refund.
3. Set the order status to "Refunded." Hive will ask you to confirm the refund by typing your username. Hive then handles the rest of the banking side of the refund.
4. Via Hon-E, notify the customer that you've processed their refund. Let them know that we cannot accept returned bees, so they may keep or release them. Apologize for the inconvenience, and include a code for 20 percent off their next order.

Additional information:

- Prior to processing any order, we confirm that the bees being ordered would not be invasive in their destination. This means it's safe for the customer to release any unwanted bees.
- We cannot accept returned bees for health reasons. We cannot risk contaminating our hives with parasites or diseases the customer's bees may have picked up somewhere.
- Bees4Less has a Satisfaction Guaranteed policy ([here's a link to it](#)). Therefore, we don't question or argue with our customers if they need a return. Instead,

we give them 20 percent off their next order. That said, we track this. If any customer abuses this policy or is rude to our staff, we will take action. We don't tolerate jerks around here.

If you have any questions about this procedure, please open a ticket for internal support (link here to internal support ticketing). Here are instructions for how to open a ticket (link to instructions).

To suggest changes to this procedure, please send a merge request to the P&P repository (link to that repo). Here are instructions for how to send a merge request (link to instructions).

Last updated: November 2023

Focus on Outcome, Not Tooling

As the team is writing the P&P, try to avoid the temptation to focus the documentation on the tools. Instead, try to focus on the function of the policy or procedure that is there to help people perform. Obviously, you can't neglect tooling in the documentation, but try to make it secondary to the outcome or goal of the P&P document. Not only are the tools likely to change—requiring that someone update the documentation—but also they're nowhere near as important as that outcome.

For instance, in [Lower Expenditures Through Intentional Use of FOSS Alternatives at Bees4Less, on page 136](#), Bees4Less management decided to switch their intranet from a fragile, custom-built content management system (CMS) to one of the many robust FOSS CMS alternatives. This is a large project on its own, but it's made larger by the need to review and revise any P&P or other documentation that refers people to the intranet. The more tightly entwined the procedures are with the previous tooling, the more effort it'll take to revise the P&P.

Since the Bees4Less P&P documentation resides in that CMS, they already needed to update it all to the new format. During that process, they refactored the P&P that there's now a section dedicated to describing the intranet and how to use its features. All the P&P then add the appropriate cross-references to this section rather than duplicating instructions for how to use the CMS tooling. In this way, they next time they have a big change in their CMS tooling, the majority of the necessary documentation changes are limited to a single section rather than scattered across the entire body of corporate P&P.

This is one way to help P&P writers focus on outcome rather than tooling, but it's far from the only one. How you approach this challenge will depend upon the tooling, P&P, and needs of your company.

Use Approachable Language

A common mistake in P&P documentation is to make it sound too formal and stodgy. These policies and procedures repel people with their drab, cumbersome phrasings, increasing the likelihood that they'll be ignored.

- Use plain language when documenting P&P.
- Avoid opaque or legalistic wording.
- Avoid passive language whenever possible.
- Don't use acronyms without first spelling out what they mean.
- Avoid jargon unless it's defined within the document.

Most companies will have a hard time with these last two points, but I encourage you to do your best to follow them. If nothing else, consider the use case of someone new to the company or team. If they can't immediately understand the contents of a P&P document, then it's likely a lot of the existing staff members can't either.

The example Bees4Less procedure in [Standardize Your P&P Format, on page 148](#), shows most of these tips in action.

Create a One-Stop P&P Shop

It can be tempting to store the security and compliance P&P with that team, the human resources P&P with them, and so forth, but I encourage you to keep all of the policies and procedures in a single location within the company intranet, wiki, or knowledge base. When someone needs a policy or procedure, they usually need it *right now*, and having a single location where they can find all P&P makes it a breeze to locate, browse, skim, and search them as needed. Also, having P&P stored in multiple locations throughout the company makes it more likely that any given policy or procedure may have multiple versions. Which one is correct? Which one should people use? Minimize the chance of this confusion by having a single, authoritative home for the entire company's P&P.

Use Version Control

For decades, programmers have used Version Control Systems (VCS) for storing different versions of their source code. Not only does a VCS like Git allow for programmers to roll back changes that don't work as expected, it also makes it easier to know which version of the code is running at any given time.

Wonderfully, Git and other VCSs aren't only for source code. All documents benefit from the features of a VCS, especially P&P. Maintaining these documents in a VCS not only makes it easier to find and change them, it implicitly

declares that the P&P will and should evolve. The versioning functionality of the VCS makes it clear when the P&P document was last changed and which version a person is using.

Furthermore, all modern VCS systems provide functionality for people to recommend changes without impacting the currently approved versions of the documents in the repository. This gives stakeholders the time and opportunity to review and discuss the proposal before it's approved. If someone makes a mistake and approves or adds a policy or procedure that they shouldn't have, these systems also come with sophisticated features for rolling back a change and viewing the differences between versions of a file.

Notice this line in the Bees4Less procedure from [Standardize Your P&P Format, on page 148](#):

To suggest changes to this procedure, please send a merge request to the P&P repository (link to that repo). Here are instructions for how to send a merge request (link to instructions).

Providing this sort of guidance in every policy and procedure encourages staff members to participate in the process and reminds them that they're trusted and empowered to suggest changes to the policies and procedures that impact their jobs. This type of call to action and encouraged engagement and discussion can become a powerful element of a positive, supportive, and productive company culture. It can even be a good first step toward implementing InnerSource, which you learned about in [InnerSource: Collaboration Culture Comes to the Company, on page 107](#).

Hire Writers and Librarians

Software undoubtedly is important to your company, so you hire specialists—programmers and IT—to take care of it. In Finance, you hire accountants to manage the books. The team over in Legal is made up of trained lawyers because this isn't something just anyone can do correctly. These are all critical functions within the company that require trained specialists to perform the work well.

As an important business asset, your P&P documentation requires the same level of specialist attention. Hire technical writers to do this work, if only on a contract or part-time basis. This is the best way to ensure that the resulting policies and procedures will meet the needs of the company. Writing is hard, and harder to do well. Entrust it to a specialist.

These documents may be well written and stored in a single location, but that doesn't mean that location is necessarily well organized for people to find

what they need. For that, you need a librarian or an information architect. Organizing information to make it easy to find and use is a profession: it's hard to do and harder to maintain.

Your company has invested a lot in its business asset of P&P documentation; don't trust the vital task of information architecture to people with good intentions but no training. A single librarian or information architect position pays for itself, increasing overall employee productivity and satisfaction by making your company's intranet and other documents useful and accessible.

Gather Knowledge and Collaborate for the Best P&P

Beyond the obvious "what do we do and when," you'll need other information for creating effective inbound FOSS policies for your company. Gathering this information in advance provides the context for the P&P. This context shapes the policies and procedures in such a way that they're more applicable and more likely to be applied correctly, and they'll require less active enforcement.

Not Everything Needs P&P...or Does It?

It can be tempting to define everything in a policy or procedure, but is it necessary? If you're in a highly regulated industry, the answer may be yes, absolutely, but for others it may not. Resist the P&P proliferation urge. Each policy and procedure brings with it its own costs for creation, enforcement, and maintenance, so you must be sure the policy or procedure is needed and that it's worth that cost.

Before starting on a new policy or procedure, make sure you review these things:

- The relevant business needs
- The company's strategic direction and plan
- The tactical requirements and constraints of the company
- Any potential risks the company may be facing

These can help guide you to decide to whether a formal P&P document is needed at all, and if it is, then they also provide that valuable context that can make the P&P more applicable.

For instance, is it necessary to create a policy that all employees must use the same set of desktop software, perhaps preventing the use of FOSS alternatives? No relevant government regulation for your industry calls for information retention requiring consistent integration across the desktop software suite. IT reports that the majority of the company uses the same set of desktop software, and the outliers provide their own support and are not a drain on IT resources.

The company's immediate and long-term strategies wouldn't benefit from creating a policy of shared desktop software. In light of all of this information, it becomes obvious that creating and enforcing such a policy would drain productivity away from more important matters. This is a policy that would hold negative business value at this point in time.

Going overboard on P&P creation brings with it additional maintenance overhead. As you'll learn in [Evolve the Policies, on page 160](#), each one of those policies or procedures must receive regular review and revision to ensure people continue to trust and use it rather than going rogue with their work-arounds. Adding unnecessary P&P adds to the maintenance overhead. This, in turn, makes the entire maintenance process look so intimidating that most people would rather not do it at all. This is a good way to undermine the good intentions of having policies and procedures in the first place.

Another problem with P&P overload is the *cognitive load* it brings for those who must use these things.¹ The more P&P people must remember, the higher the chance that mistakes will creep in. The P&P then have the opposite effect to that intended. Rather than helping people do the right things for the right reasons, instead their brains are so burdened by often conflicting instructions that they expend as much energy on parsing the P&P as on performing the task they intended to do.

So be careful when creating your P&P, and make sure any new additions are necessary. However, don't avoid adding necessary P&P simply because you're afraid of increasing the collection. It's a balancing game as well as an ongoing effort, but it's one that's worthwhile.

Collaborate on P&P Development

Policies and procedures are neither formed nor enforced in a vacuum. Collaborative development of P&P, including all stakeholders, will help ensure that people don't ignore or work around them.

As you learned in [Consult the People Who Will Use the P&P, on page 148](#), regardless of whether it's documented, people will have their own P&P for getting their work done. Take these into account when creating a formal policy or procedure. By doing so, you not only end up with higher quality P&P—because they've already been tested—you also have P&P that are easier for people to adopt. Also, consider whom new P&P will affect and be sure to consult each stakeholder.

1. https://en.wikipedia.org/wiki/Cognitive_load

In the desktop software example in [Not Everything Needs P&P...or Does It?, on page 153](#), P&P for desktop software affects HR differently than it affects IT. Take all the relevant use cases into account, and you'll have P&P that people are more likely to understand and use.

Deploy Policies Effectively

On the surface, deploying policies and procedures may seem like a simple matter, but the way in which it's handled can make or break any P&P effort within your company.

You may be inclined to announce new P&P as soon as they're ready, then brush off your hands and walk away feeling satisfied in a job well done. It's not that easy. Few things irritate people like running into new and unexpected walls of policies and procedures. It disrupts their workflow and erodes their trust in the company. Some people will immediately look for and start using work-arounds, and from there on out it'll be a struggle for you to get people to pay attention to the P&P on which you spent so much time.

You've already learned one way to prevent this problem: collaboration. Including stakeholders in the P&P development process proves to people that their input is valid and actively desired. People thus included end up with a personal investment in the P&P and their successful implementation. These people become allies and ambassadors and are priceless.

Communicate, Communicate, Communicate

Collaboration also helps with the next P&P deployment step: communication. Policies and procedures don't just appear, springing fully formed like Athena from the head of Zeus. The P&P aren't a surprise to the people working on their creation, so they shouldn't be a surprise to anyone to whom they'll apply. Communicate before, during, and after P&P deployment so people are able to prepare and make necessary modifications to their workflow.

This communication shouldn't only include the what of the policies and procedures but also (and especially) the *why*. Your company has hired smart people; they can handle the why of a policy or procedure, and they deserve it. Sharing the reasoning behind P&P shows you trust people, which in turn builds employee trust in the company. Don't assume that the why of P&P is evident to everyone.

Remember, you may have been working on the policies and procedures for a while. For you, this is all old hat. For others, it'll be the first time they hear of these things. Set them up for success by providing them the information

they need to understand this new way of working, and you'll also be making the P&P easier to follow and adopt.²

Don't Forget Training and Onboarding

Training is a step that many companies overlook when deploying inbound FOSS policies and procedures. Frequently, they only remember the need for training after P&P have already gone live and people are struggling to use them. The company has now unwittingly fostered ill-will toward the P&P, decreasing the likelihood that people will follow or enforce the P&P correctly or at all.

Create any necessary training before deploying new P&P and have it ready to roll on deployment day. When creating that training, remember that deployment day is only the start of the P&P life cycle. New employees may need to learn these policies and procedures, so work to ensure that the training becomes a part of the onboarding process.

Enforce the P&P

A policy or procedure that no one enforces or uses is noise; it's cruft that collects in the corners of the organization, piling up. It's a mental burden: "Should I use this policy when no one else is? Should I enforce it? What do I do?" The company's staff have better things to do with their brains than second guess the validity of P&P. If you're going to take the time to create a policy or procedure, make sure you can also take the time to enforce it.

Try to Automate Enforcement

Whenever possible, look for ways to automate the enforcement of any P&P. Not only does this make it easier for everyone to do the right thing, it also means the enforcement happens equally and objectively across all to whom the P&P apply.

For instance, if you establish a policy that all programmers must use inbound FOSS components that are compatible with the GPL-3.0 license, you can work with your team to add automation to verify that the license of any newly added dependencies match those that the compatibility chart in [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#), shows are compatible. The automation runs every time someone commits new code and starts a software build, so incompatible dependencies are discovered before the software ever ships to customers.

2. <https://hbr.org/2020/05/good-leadership-is-about-communicating-why>

Beware, though, of the trap of automation. A lot of P&P enforcement can occur via automation but not all of it; some requires people to apply their judgment. Another danger is over-automating, investing more time in automating simple P&P than it would take to do the enforcement manually. So when diving into a project to automate P&P enforcement, start with those P&P that are easy to automate and those that would present a risk to the company if their enforcement were missed.

Following are some examples of policies that you may be able to automate:

- Dependencies without an obvious copyright/licensing header require additional review.
- Dependencies without an easily located license require additional review.
- Do not use dependencies that are incompatible with the GPL-3.0-or-later license.
- Always make sure the latest shipped source code and build scripts are publicly available.
- Always keep a copy of software dependencies in the local archive.

You'll learn more about why you might want policies like these in [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#), and [Chapter 12, Maintain FOSS License Compliance, on page 201](#).

Provide Empathetic Enforcement

The *e* in *enforce* stands for “empathy.” OK, no, it doesn’t, but empathy should be the most commonly used tool in your enforcement tool chest. The chances that someone is willfully transgressing a policy or procedure are slim. It’s much more likely that they don’t understand the P&P or that they don’t even know it exists.

Provide correction and coaching to ensure the employee understands the problematic policy or procedure, but then look elsewhere for the source of the error. Rarely is this ignorance the fault of the employee; it’s a breakdown in the P&P process. The transgressing employee has done you a favor by finding a bug in the system. Some step somewhere didn’t work: perhaps the P&P are poorly written, perhaps the communication or training plans don’t work, or maybe the P&P are out-of-date or otherwise invalid. Whatever the reason, take the time to discover and fix it. If you don’t, people will start to lose faith in the policies, procedures, and management.

Be Prepared with an Escalation Path

As much as we may prefer otherwise, things don't always go to plan. Your policies and procedures may seem OK now, but once people start to use them, conflicting ideas about how things should be done may come up. Even with the most empathetic enforcement techniques, if you don't give people a way to express their conflicting P&P ideas constructively, you may end up having to deal with confusion, frustration, and maybe even anger.

If you've followed the advice of [Standardize Your P&P Format, on page 148](#), your P&P will include a contact method for when people have questions. Sometimes that's not enough, and those questions need more or different attention. Consider creating and documenting an escalation path that people can follow to report problems with the P&P.

For example, Seoyun is trying to use Bees4Less's expense reporting policy, which requires she submit paper receipts. A number of her expenses came from online purchases. Does this mean she must print out those order confirmation emails, or can she send PDFs to the finance team instead? The escalation path for such a P&P question may look like this:

P&P Support ticket → Seoyun's manager → Seoyun's department head → Finance department head → Bees4Less ombudsperson

Being prepared with an escalation path for P&P matters can resolve a lot of conflicts even before they get started. By establishing and documenting escalation paths and mechanisms for sharing feedback, your company also shows employees that their opinions are valuable and welcome.

Trust People with the Authority to Do Their Jobs

A common problem with P&P enforcement is companies asking employees to enforce certain policies but not giving them the authority to do so—for instance, asking someone to enforce the policy, mentioned previously, of GPL-3.0 compatibility but then not giving them the authority to stop a software deployment if they find discrepancies. This is worse than having no policy at all, as it creates ill will on all sides and can potentially put the company at risk.

Always ensure that the people tasked with P&P enforcement have the authority to do so, and don't overrule them if they justly use that authority. Think of it like the chief medical officer on *Star Trek*: even the captain has to listen to them in medical matters.

Use Checklists

Checklists are vital in transportation and healthcare, saving lives every day. They may not save lives in your company, but they certainly can reduce the cognitive burden of having to remember all of the steps necessary to remain in compliance with P&P.

Checklists can be valuable when enforcing P&P and also can help to guide any automation efforts. Establish a culture and process for launch checklists across the company. No product or project should ever launch without passing all of the criteria on the relevant checklist, and some of those criteria are the P&P that apply in that situation.

For example, if your company has a policy of using only Creative Commons-licensed assets in its marketing material, the checklist for releasing a new marketing campaign might include things like these:

- ☐ Make sure all assets are in the Marketing version control repository
- ☐ Ensure all external assets are under Creative Commons licenses
- ☐ Ensure no external assets are under a Non-Commercial (NC) Creative Commons license
- ☐ Open a ticket for the web team to update the attributions page on the website

Naturally, if your company adopts checklists to aid in following P&P, make sure that those checklists and their related P&P get stored in the same location. Following the advice of maintaining a centralized repository of P&P but then allowing the checklists to scatter all over the company's intranet is a good way to frustrate employees, leading to P&P avoidance. Checklists are yet another product in that one-stop P&P shop. You can learn all about how to use checklists effectively in your business from Atul Gawande's book, *The Checklist Manifesto*.³

3. <http://atulgawande.com/book/the-checklist-manifesto/>

Evolve the Policies

The world around us, your customers' needs, your entire market and product space, and everything else is constantly evolving. Why, then, do most P&P seem carved in stone? Considering the ever-changing nature of business, P&P should instead be carved in something much less permanent—Jello-O, perhaps.

In the movie *Glengarry Glen Ross*, the character Blake famously tells the other salespeople to pay attention to their ABCs: Always Be Closing. For policies and procedures, a better version is *Always Be Changing*. An out-of-date policy or procedure is an ignored policy or procedure, and that puts your company, your business, and perhaps even your customers at risk.

Commonly, P&P maintenance is an ignored step in the P&P life cycle. Since you're establishing P&P anyway, make part of that work the definition of meta-P&P: policies and procedures about the policies and procedures. Some of the things you should consider when discussing P&P maintenance are these:

- How regularly should P&P be reviewed?
- What's the process for that?
- Who is responsible for the changes and the ensuing communication?

Defining this sort of thing up front can save you a lot of headaches (and meetings...which you may see as the same thing) down the line.

One way to always be changing is to crowdsource the iteration of P&P. By using a version control system, or by making it simple to report updates and corrections to policies and procedures, you can develop a culture where everyone feels ownership for the P&P that apply to their work. A single-click “report a problem” link on every policy or procedure can go a long way to shining light on problems that might otherwise cause people to stop trusting or using P&P.

People who are new to the company or to their role will have a valuable perspective here. Acknowledging this and making it both easy and acceptable for them to provide feedback on P&P will help improve the work lives of everyone who comes after them.

Clean Up the Clutter

Sometimes you'll find that what's needed is to get rid of a policy or procedure altogether. It's surprisingly satisfying: cleaning up the collection of P&P and removing the overhead of having to maintain P&P that no longer apply.

If you do remove a P&P element, make sure you clean it up completely. Redirect the page to a new or updated policy, and remove traces of it from automation, training, onboarding materials, and anywhere else it may be represented. Also, make sure you communicate this removal well! It doesn't help to clean up a policy or procedure if people don't hear about it and continue using it in their day-to-day work or enforcing it after it's no longer necessary.

Now that you have a better idea of how to handle policies and procedures correctly, let's learn about some of the P&P that you may need to support your inbound FOSS strategy.

Get Started with Your Inbound Policies

In [Chapter 9, Establish Effective Inbound Policies and Procedures, on page 145](#), you learned that creating and enforcing useful—and used—policies and procedures (P&P) isn't the matter of a moment. It requires time, collaboration, communication, and empathy to have policies that will protect the company and business effectively. You won't see any of the benefits ([Chapter 5, Strengthen Your Business Through FOSS, on page 87](#)) and will experience many of the risks ([Chapter 4, Avoid Common FOSS Business Risks, on page 65](#)) if the company does such a poor job of creating and enforcing its FOSS policies and procedures that people find work-arounds rather than follow P&P.

In this chapter you'll learn about some of the inbound FOSS policies and procedures your company should consider and also how to make all of your policies and procedures more useful, relevant, and viable for all staff.

Inbound FOSS IP Policies

The most common of all inbound FOSS policies is intellectual property (IP). If your company has any policy at all around inbound free and open source software, it's probably because of IP concerns. An IP policy is intended to respect the license terms and the IP of the authors of FOSS projects, but it's also there for the protection of the company's own intellectual property.

The IP policy will require that all inbound FOSS be tracked and logged in some way. That log should include the name of the software, its license or licenses, where to find the source code for the project, and how the project is going to be used in the company. Often, the policy states that no FOSS can be used until it's been approved by company authorities, usually the Legal department.

Know the Stakeholders

Who is the audience for the inbound FOSS P&P? When you think about it, everyone in the company uses software, but that doesn't mean a given policy or procedure necessarily applies to everyone. Always keep the audience front and center, and you're more likely to end up with P&P that people will take to heart.

Legal

The primary stakeholders for inbound FOSS policies are Legal and Software Development, but, depending upon your company's needs, other departments have stake in IP matters as well. As previously mentioned, the team from Legal will be intimately involved, often not only approving use of inbound FOSS projects but also creating and enforcing the inbound FOSS IP policy in the first place.

Software Developers

Software developers are the ones at the sharp end of many inbound FOSS policies. Commonly, developers will balk at having to log all inbound FOSS components and then wait for approval before they can use a component, claiming that it slows them down and delays delivery of products to customers. All too often this claim proves correct if the policy enforcers aren't adequately staffed or funded to respond rapidly.

Others

A common theme of this book is "FOSS can have company-wide impact." Keep this in mind as you're considering the audience for your P&P, and consider the full scope of whom the policies may affect. Recall from [Use Checklists, on page 159](#), for instance, when Marketing created a checklist to ensure proper handling of Creative Commons–licensed assets. Don't assume you already know the stakeholders as you start your inbound FOSS P&P journey. Pause and take a little time to confirm (or disprove) your assumptions.

Consider the Risk Profile

What is the risk profile for your company and its business? That is to say, what are the risks it faces and what additional risks can it afford to take on? Each company has a unique risk profile, and the nature of those profiles vary a lot based upon industry. The financial sector has one flavor of risk profile, medical another, entertainment and manufacturing each their own flavors, and so on.

When considering inbound FOSS P&P, it's critical to take the company's risk profile into account. Does it have a low risk profile or a high one?

- *High risk profile:* The company is able to take more risks without much damage to its brand or its customers. A software startup may be able to get away with unfettered inbound FOSS and therefore have loose P&P and oversight. This company either has a high risk profile or is willing to take on a higher level of risk to help it move toward its strategic goals.
- *Low risk profile:* The company can take few to no risks, as any risk could bring outsized negative effects. A medical device manufacturer can't play so fast and loose with its inbound FOSS P&P. Not only is it more difficult to update software in a medical device, but also people's lives are on the line here. That's not a risk any company should take lightly.

Awareness of your company's risk profile will allow you to craft inbound P&P that match the risk the company is willing and able to take on at that time.

Automate It

Depending upon the system used for logging the components, the approval process may be automated. It's even possible to add the component-logging step into the automated software test and build system, removing the need for a programmer to log newly added FOSS components manually. These forms of inbound FOSS IP policy automation can be time-saving, but they also add complexity and overhead to business operations. Before diving into an inbound FOSS IP policy automation project, make sure you know what you're getting into. These sorts of automations need regular maintenance and may require the company to purchase or license software from external vendors to support the logging and approval processes.

To ease the automation process, the legal team can establish an `_approve list_` of FOSS licenses. The automation can automatically and instantly approve any logged FOSS components that use licenses on this list. For example, if the product the team is working on is licensed under GPL-2.0-only, according to the compatibility matrix from [FOSS License Compatibility, on page 61](#), these FOSS licenses could be on an approve list:

- MIT
- BSD-3-Clause
- BSD-2-Clause
- MPL-2.0
- LGPL-2.1-only
- LGPL-2.1-or-later

Conversely, Legal could instead establish a *stop list* of FOSS licenses. For the same GPL-2.0-only situation, a stop list could include:

- Apache-2.0
- LGPL-3.0-only
- LGPL-3.0-or-later
- GPL-3.0-only
- GPL-3.0-or-later

Such lists are common, as are related lists for *approved for internal use only* for software that will never be distributed outside the company and *disapproved at all times without VP signoff*. Using these lists as filters speeds up the process of applying the IP policy and can make it easier to automate.

Don't Fear Reciprocal Licenses

Decades ago, near the start of the Free Software and open source movements, many large enterprises became paranoid and outspoken about what they perceived as the dangers of reciprocal (copyleft) FOSS licenses. They were convinced that allowing reciprocally licensed software into their company would result in the loss of their intellectual property as they were “forced” to release it to the world for free by the terms of the licenses. As a result, their inbound FOSS IP policies banned nearly all reciprocally licensed software, and their outspokenness established an industry-wide fashion that was followed by smaller companies.

Despite decades of disproving this paranoia, the perception of IP risk with reciprocal software remains, as do overly restrictive inbound FOSS IP policies. Remember from [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#), FOSS license terms—even reciprocal—trigger on the *distribution* of products that use the components, not on use. It also matters whether the software is a derivative or whether it's *merely aggregated*.

When viewed logically, based on the facts of the licenses, there are few actual risks in using reciprocally licensed FOSS components in your company's products. However, regardless of the type of license—reciprocal FOSS, permissive FOSS, or proprietary—your company is always subject to the license terms and must always ensure it obeys them for any software in its supply chain.

Inbound FOSS Security Policies

Thankfully, it's becoming a lot more common to see companies creating and enforcing security policies for inbound FOSS software. I say “thankfully” not because FOSS is inherently insecure—you learned in [Chapter 5, Strengthen](#)

[Your Business Through FOSS, on page 87](#), that it's not—but because people's *handling* of FOSS is. You'll learn a lot more about this in [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#), but here are the basics.

All software has bugs, and some of those bugs may have implications for security. This is a fact of software, and one that many people prefer to ignore. Updating (patching) software to have the latest and most secure version can be an inconvenience, but it's not nearly as inconvenient as having the company systems compromised and customer data stolen because of an unpatched bug.

Most software packages, FOSS or proprietary, now try to make it relatively easy to update to newer versions. Yet how can your company do that if it doesn't know which versions it's running, whether those versions are out-of-date, or whether the software is being used at all?

The same processes used to log FOSS components for IP policy purposes also come into play for inbound FOSS security policies. It's impossible for your company to stay on top of software security updates if it has no idea what software is in use.

For example, in 2021 security researchers discovered a critical security vulnerability in the popular FOSS logging component log4j; they named the vulnerability Log4Shell. The log4j FOSS community fixed the vulnerability nearly immediately, but even now, years later, researchers say that only 50 percent of the affected software has been patched with the fix.¹ It's unlikely that those who maintain or run this unpatched 50 percent are even aware that they're running vulnerable software and are targets for criminals.

To prevent this from happening to your company, create a policy requiring the logging of all FOSS components in use, and create a procedure to check component versions regularly, ensuring they're the most up-to-date and secure.

As you're considering these inbound FOSS security P&P, don't forget any software or tools that your business has sent out into the field or the market. Before distributing this sort of software, either on hardware your company sells or software it licenses to customers, make sure you know how you'll distribute security fixes to them. Just because the software has left the confines of your company doesn't mean you're no longer liable for maintaining it. You owe it to your customers to provide them with secure products.

1. <https://www.darkreading.com/application-security/one-year-later-log4shell-exposed-attack>

Plan for Disaster

What a delightfully boring world it would be if nothing ever went wrong. Alas, that's not the world we live in, even though you'd never know by looking at many companies' inbound FOSS policies.

It's tempting (and a lot easier) to follow the Happy Path when creating these P&P, so many companies take this easy way out. Everything is great and will always be great and, gosh, isn't that great—until it isn't. Only then do these Happy Path walkers discover that they're strolling among houses of cards that are now collapsing all around them.

Often, these companies were warned of their danger well in advance but neglected to listen to or take action on the warnings. These latter-day Cassandra^a usually report into the companies' security teams and do whatever they can to prevent damage from security incidents, whatever form they may take. If they can't prevent the damage, they usually have ideas about how to make the company more resilient so the damage has less impact.

When creating these P&P, specifically ask your security team to come up with some inbound FOSS disaster scenarios as well as some related mitigation plans. Some of the scenarios you're likely to see include these:

- A critical FOSS component contains an actively exploited vulnerability (à la Log4Shell).
- A FOSS maintainer sabotages a component that your company uses.
- A criminal successfully uses social engineering to add a Trojan horse to a component.

Include your security team in all your FOSS P&P discussions, and your company will be prepared should it ever stray from that Happy Path.

a. <https://en.wikipedia.org/wiki/Cassandra>

Inbound FOSS Sustainability Policies

A commonly overlooked category of inbound FOSS policy is *sustainability*. You'll learn all about FOSS sustainability in [Chapter 13, Sustain. Maintain. Retain., on page 213](#), but to get you started, here are some things to consider with regard to policies supporting the sustainability of your company's inbound FOSS.

Let's say you're going on a long road trip. You and your best friend are going to drive the back roads across the country. Before you set out, you need to make sure that your car can make the trip, so you take it to the mechanic

for a tune-up, oil change, and tire rotation. You now feel more assured that this vehicle will get you where you need to go with minimal problems.

The software in your company helps to get it where it needs to go in its business matters. If that software—or a critical component of it—breaks down, then your company is going to be in an unpleasant situation. It may even lose customers if the breakdown is bad enough. Like with your car, your company can take steps to ensure that all of the software components on which it relies remain in working order.

For example, you can create a policy about selecting FOSS components based upon the sustainability of the project:

- Is the project actively maintained?
- Does it have a healthy community, or is it plagued by trolls and rudeness?
- Does the project have enough maintainers that none of them are likely to burn out, or does it instead have one or only a few maintainers?
- Are there support options to which your company can subscribe, either from the project itself or from third-party vendors?

These are a few of the criteria that you can use to determine whether a project is sustainable, which is to say, that it'll be around and healthy for as long as your company needs it.

Another policy your company may want to consider is to perform regular reviews of its inbound FOSS components and then provide monetary or other support to those that are most critical to its business operations. Contributing code, documentation, security or accessibility reviews, or even infrastructure as a matter of company policy can help those critical projects remain sustainable. You'll learn more about how your company can contribute to FOSS projects in [Chapter 17, Master Contribution Fundamentals, on page 263](#), and more about providing support to projects in [Chapter 13, Sustain. Maintain. Retain., on page 213](#).

Inbound FOSS Integration Policies: Vending

Some companies, when bringing inbound FOSS components into their software, have a policy requiring that those inbound components be downloaded and stored locally.

The practice of maintaining and using copies of the third-party software used by the company is known as *vending*. This vendored software can be committed to the company's version control system (usually in a `/vendor` or `/third-party`

directory), kept in an internal container registry, or in some other dedicated artifact repository. When the company builds its software, it then uses these copies of the dependencies rather than going out and retrieving the latest versions from the internet (also known as *upstream versions* of the component).

This may sound like a pain, but advantages to vendoring inbound FOSS in this way largely involve avoiding surprises and risk for the software the company creates. Vendoring an inbound FOSS component then *pins* that component to a specific version. From there on out, internal software development will use only that version of the FOSS component until the company makes the choice to update to a new one.

Vendoring the component makes it a known quantity: the software development team always knows what features are available to them and that the chosen version of the component will work in the software build. The legal team knows the license of the chosen component and doesn't need to worry about it changing with future versions (an uncommon but very real risk that often goes overlooked).

Building software from vendored components can reduce security risks: if a new vulnerability sneaks into the upstream version, it won't also sneak into your software. Relatedly, if a FOSS maintainer sabotages their own component (another rare but real risk for which we've seen several examples in the past few years), a vendored version of that component won't be affected.

Finally, vendoring inbound FOSS components creates a cache from which all internal software builds can draw, reducing network latency for the builds. All internally developed software can use the same repository of vendored components, creating consistency across software development projects.

Vendoring Isn't Free

For all the potential advantages of vendoring inbound FOSS components, it's not the right choice for every company. It's a complex process. It's not as simple as creating a shared drive on the company intranet then dumping components in there.

Vendoring isn't free; it comes with high costs for maintenance and infrastructure. Component proliferation, where Team Cat needs version 1.5 of a component but Team Dog needs the features available in version 1.9, is a problem requiring vendoring of both versions.

More troublesome is the problem of allowing components in the vendoring repository to go stale. Review of vendored inbound FOSS components must

become a regular duty of someone within your organization, most likely in your OSPO ([Chapter 6, Introducing the Open Source Program Office, on page 101](#)). They should remove old or unused components and update to more secure versions of the remaining ones, all the while coordinating with the development teams and other stakeholders to prevent problems with the software build or quality.

So think carefully before creating a policy to vendor all inbound FOSS components, and weigh the benefits and disadvantages for your company's specific requirements.

Other Policies You May Need to Consider

It's tempting to look at the policies listed here, draft your own versions of them, and then dust your hands off and walk away thinking all your inbound FOSS policy problems are now sorted. The policies mentioned in this chapter are the most common ones for inbound FOSS, but they're far from the only ones possible. Depending upon the requirements of your company, other policies may be needed. Some examples include the following:

- Never maintain internal forks of FOSS projects.
- Always contribute code fixes to the upstream project (see [Chapter 17, Master Contribution Fundamentals, on page 263](#)).
- When selecting FOSS components, first check the approve list for components already used by the company.
- When looking for new software, first look for FOSS options.

For example, the General Services Administration (GSA) of the United States government has an “open first” policy: when it looks for external software, API, or data options for a new program or initiative, it must first look to free and open source for those options.² Many companies find that a similar open-first policy serves their business needs.

Companies that are especially sensitive to potential leaks of their intellectual property may feel the need to have a policy that it's entirely fine to use free and open source software components but *not* if those components are maintained by direct competitors. This can be difficult to enforce, but it hasn't stopped companies from trying.

A variable that may influence what inbound FOSS policies your company needs is its *software supply chain*. Let's learn more about it.

2. <https://open.gsa.gov/oss-policy/>

Know the Links in Your Software Supply Chain

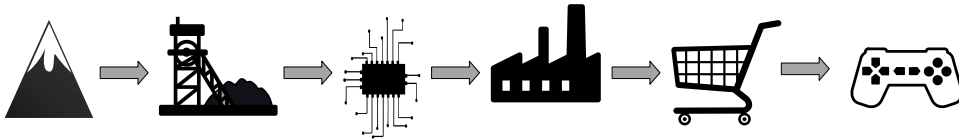
Software supply chains are complex and—perhaps not coincidentally—regularly ignored by companies of all types. For them, their software supply chain is out of sight and out of mind. Companies have largely been able to get away with that to this point, but times they are a-changin’. More and more often, companies are learning that ignoring and neglecting their software supply chains can be existentially bad for business. Thankfully, it’s never too late to start where software supply chain awareness is concerned. In this chapter you’ll learn what your business’s software supply chain (SSC) is, how an SSC forms, how to learn about the links that compose your SSC, and what to do with that information.

Before we get started on this topic, it’s fair to warn you that it’s a big one. What you’ll learn here will lay the foundation on which you can build going forward and allow you to make informed judgments about which resources and approaches are right for your business, company, and customers.

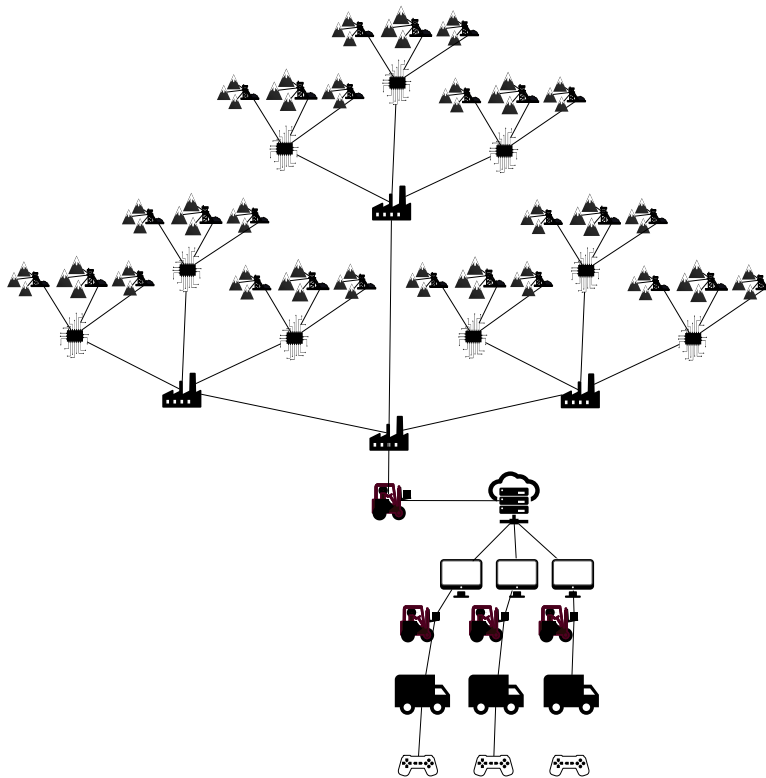
Supply Chain Defined

Supply chains are everywhere and touch upon pretty much everything in our modern lives. For instance, let’s consider a video game console.

Ore is extracted from a mine and then is sent to a refinery. The resulting material goes to a chip fabricator, who sends the finished computing chips to a factory to be assembled into a game console. The console then heads to a retailer, where you buy it and bring it home. Congrats on your shiny new video game console! This chain is depicted in the [figure on page 174](#).



This example is obviously simplified but still portrays several links in this supply chain. Upon closer inspection, you'll find that every supply chain, including this one, is actually a supply web: each link in a chain has its own complex supply chain as well, as shown in the next figure:



As with real, physical chains, breaking a single link causes the entire chain to collapse (no video games for you). Unlike real, physical chains, the hidden web nature of supply chains means that a single broken link can impact far more than a single chain. This makes supply chains especially fragile. Entire industries exist to observe, analyze, and maintain our supply chains, ensuring you can continue to get your hands on the latest video game console.

Supply webs, like any complex system, can quickly get overwhelming. It's useful to be aware of the extent of the web but may not be as useful to *act*

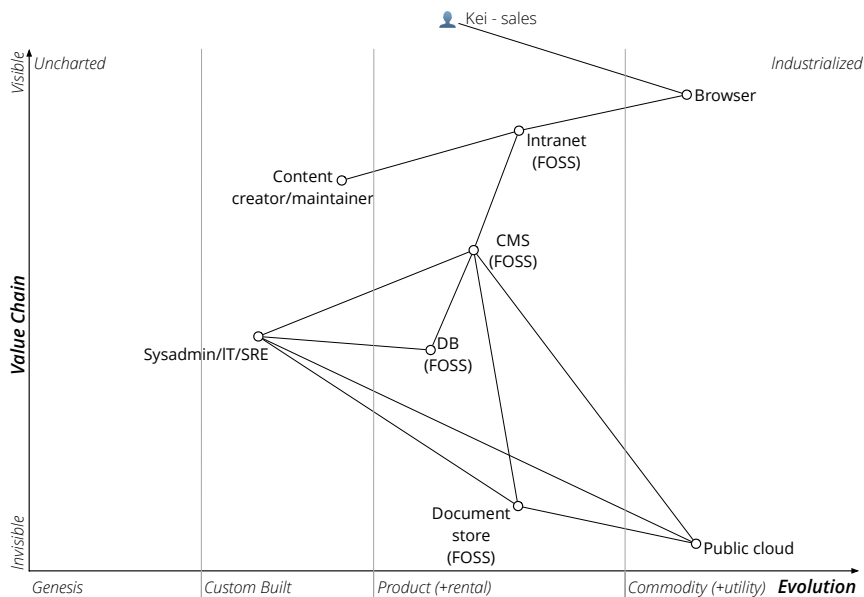
upon it. When considering your company's supply chains and webs, do so in the context that's useful for the matter at hand:

- Define the problem you're looking to solve or question to answer for yourself, your team, your company, and your business.
- Set the boundaries and parameters for your supply chain studies accordingly.
- Be aware of the bigger supply web picture for those times when your problem approaches the edges of your chain.

Keeping context in mind will give you the knowledge and perspective you need to make better-informed decisions, including the decision whether to expand the supply chain definition used in day-to-day operations.

The Software Supply Chain

Your business has physical supply chains in some form, but it also has *software supply chains* (SSC). You first learned about the software supply chain in [Chapter 7, Recognize and Acknowledge Your Inbound FOSS, on page 123](#). It has many links, and each one is a different piece of software that's critical for the functioning of your business and delivering value to your customers. For example, recall the content management system (CMS) example from [Chapter 8, Inbound FOSS in Your Corporate Strategy, on page 131](#):



You can see the software supply chain in action here:

- Kei uses their browser to access the intranet.
- The intranet is built upon a CMS.
- The CMS uses a document store and is running on public cloud infrastructure.

Without every link in this SSC functioning as expected, Kei can't access the sales material necessary to close deals.

Again, this supply chain is highly simplified and hides a lot of complexity. Each link has its own SSC. For instance, let's say your company chose the Free Software package Plone as the CMS link in this chain. Version 6.0 of the Plone software has an SSC of eight direct links (components on which it's directly dependent) and an indirect software supply web of 228 links (components on which its direct components are dependent).¹ That's a lot of links, and even so it's still only a fraction of the software involved in running an instance of any CMS software. The specific links in the chain will vary by CMS software chosen, but the overall complexity will not.

Types of FOSS Acquisition

In the preceding example, the company chose to use the Plone CMS after evaluating a number of options. Finding Plone fit their needs best, they made an *active choice* as an organization to commit the resources necessary to install, operate, and maintain this piece of software. This is only one of the many ways that software can enter your SSC. Knowing the other ways will help you as you start investigating your own SSC and becoming familiar with its many links.

With proprietary software, its acquisition follows a fairly well-known process. Often a salesperson is involved, with the requisite negotiations, contracts, and purchase orders. This is the standard vendor relationship with which your company is already familiar. FOSS, on the other hand, rarely has such a clear-cut and well-tracked path to acquisition. Instead, you'll find two primary methods by which FOSS finds its way into the SSC of your business.

Intentional

The first method is the *intentional acquisition* of FOSS. Similar to proprietary software acquisition, someone in your company takes the time to research

1. <https://deps.dev/py/pi/plone>

and compare the options for FOSS components and applications. They then make a selection, implement the software, and support its use in whatever ways are necessary. The intentionality stems from a need to architect and maintain the software solution. In the example in [The Software Supply Chain, on page 175](#), Bees4Less made an intentional decision to select a FOSS CMS.

This much effort usually comes with the implication or requirement that someone track the FOSS in some way so they can maintain it and apply updates when they become available. Of course, this is a fairly idealized view of how things end up working in the real world. Often it's not necessary to go through a formal process to select, implement, and track a piece of FOSS brought into your software supply chain, but some sort of process is usually necessary when that acquisition is intentional.

Invisible

The other method for FOSS acquisition is the *ad hoc* or *invisible* one, and, for better or worse (usually worse), it's the much more common of the two. In this situation, a person needs to get their work done. They go to the internet and locate a FOSS tool or component that will allow them to do that, then bring it into the company's software supply chain. Typically, this new link in the chain appears silently, with no recordkeeping, tracking, or maintenance. Effectively, that link in the chain is now invisible, which adds risk to your business.

This phenomenon—which isn't exclusive to FOSS—is known as *shadow IT*. While it's possible that *any* link in your SSC could fail at any time, should an invisible or shadow link break or be compromised, it's unlikely your company will know it until it's too late. The *Log4Shell* example from [Inbound FOSS Security Policies, on page 166](#), is an example of a broken link that's still invisible to many companies.

Vendors Maintain, but Maintainers Aren't Vendors

When starting to investigate and manage your free and open source software supply chain, it's tempting to fall into the habits and mindset that you've developed by working with and managing proprietary software, including believing that the source of the software is a vendor whom you can contact whenever you need assistance. These are habits you'll need to break.

From the business perspective, there certainly are similarities between proprietary and FOSS components in your SSC. It's all software that your company relies upon. It all comes from sources outside of your organization. Your company needs to be aware of all the links in its SSC, regardless of their source.

For all that, proprietary links in your SSC usually come with licensing fees, support contracts, and other contractual obligations on the part of the vendor. If something goes wrong with the software, your company can complain to the vendor. They can provide necessary maintenance to ensure the software continues to work as expected. Furthermore, with that contract in place you likely have legal recourse should the vendor be unwilling or unable to meet their obligations.

Not so with the FOSS links. Unless your company happens to employ the maintainers of a FOSS component, you're unlikely to have any monetary or legal relationship with the community who creates and supports that FOSS. Your company is free to request assistance, but any help is at the whim and availability of the FOSS community. Whereas a vendor has paid staff to help support their software product, most maintainers are doing the work for free and independently. If your company has not shown any willingness to support that work and the FOSS community, and if it instead appears to be exploiting the work of volunteers, then you're unlikely to receive rapid (or any) help from them.

Remember also that most FOSS licenses include some form of disclaimer of warranty (you learned about these in [Liability, on page 70](#)). These disclaimers essentially mean you have little legal recourse like what you would have with proprietary vendors. Your company does, however, have the freedom to view and modify the source code for themselves rather than wait for someone else to resolve any issues. That's something you're unlikely to get from a proprietary vendor.

Software Supply Chain Awareness Is Growing

Because the ad hoc method of FOSS acquisition is so common, most companies have a surprisingly large number of invisible links in their software supply chains. It's easy (and somewhat valid) to point an accusatory finger at programmers, scolding them for introducing those invisible links, but companies should also turn that finger on themselves. For too long, companies have overlooked their software supply chains, taking them for granted and, for the most part, lacking awareness that an SSC exists or is something that requires maintenance. If a company is setting an example of disregarding its SSC, it can hardly blame the programmers it employs if they follow suit.

This is starting to change, largely forced by external issues. These issues typically take the form of violent acts performed on links in the software supply chain. Some of these acts, such as the Heartbleed vulnerability in the critical OpenSSL cryptography toolkit² or the Log4Shell vulnerability in the popular log4j logging

2. <https://en.wikipedia.org/wiki/Heartbleed>

library,³ take advantage of software flaws to gain illegal access to systems and data.

Others stem from the creators of the FOSS components themselves. Usually as a form of protest, some creators have mutilated or completely removed the source code for the components they maintain,⁴ effectively and swiftly breaking every software supply chain of which the components are links. Thankfully, this second type of act is rare, but it *does* happen, and it causes a big mess each time. Far more common, unfortunately, are the criminal acts that exploit vulnerabilities in links in the SSC to steal data or hold companies at ransom.

These criminal acts have become such a problem that they've led to the third type of external act that's forcing the issue of software supply chain awareness: government intervention. World leaders recognize the risks posed by attacks on links in the global software supply chain and are taking action. One example of this is the U.S. Executive Order on Improving the Nation's Cybersecurity (EO 14028).⁵ Released by the White House in May of 2021, EO 14028 aims to improve the cybersecurity of the United States. This executive order lit a fire under many companies to start paying attention to their software supply chains, discovering all of those invisible links before they break.

Thanks to these three types of external acts, whether they like it or not companies are starting to become aware (some painfully so) that they have software supply chains. They may not yet know what links are in their specific SSC or what to do to maintain either their links or their chain, but knowing that the SSC exists at all is still a big step in the right direction. The next step is to learn more about that chain, identifying the links and inspecting them to see whether any of them are weak.

How to Learn About Your Software Supply Chain

When looking to learn about your software supply chain, the key acronym to know is SCA. Officially, this stands for “software composition analysis,” but it may as well stand for “supply chain analysis” since, effectively, that's what it is. SCA is the process of analyzing a piece of software to determine various data about it, including its dependencies (components), the licenses associated with each component, the version of the compo-

3. <https://en.wikipedia.org/wiki/Log4Shell>

4. https://en.wikipedia.org/wiki/Npm#Compromised_and_disruptively-edited_packages

5. <https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/>

ment, and any security concerns or vulnerabilities reported for each component. SCA can include the entire SSC for a given piece of software: FOSS and proprietary components, dependencies of dependencies, and every scrap of software involved in any way (the complete supply web), or it may only include immediate first-order dependencies (the immediate supply chain).

Depending on your business needs though, instead of a detailed picture you may only need an overview. With SCA, you get to define how far out into the branches of the dependency tree you wish to go. For some companies, their resources and risk profile allow them to get away with only a first-order view of their SSC: the links in the chain that they directly use. Other companies—especially those in highly regulated industries—require that second-, third-, or more-order view of their SSC. They need to see the complete software supply *web*, not simply the most immediate *chain*.

Manual Research

It's possible to do this sort of thing manually; I know of some Fortune 500 companies that approach the problem that way. This manual method of SCA relies on software developers, IT, and anyone else who uses or brings FOSS into the software supply chain to submit the necessary information via a form or application or even simply add it to a shared spreadsheet. This method has a low barrier to entry, so it's a good place to start if your company has never looked at its SSC before. It can at least give you an idea of what you're facing as far as gaining awareness of your SSC. The method also can be effective for very small software projects.

Overall though, it's not a good long-term solution. The manual nature of the process means it's prone to human error, both in the data submitted and in submitting it at all. The technical people at the submitting side of the equation see the method as drudgery or busywork, so submitting the data becomes resented, sidelined, and forgotten. When this happens, the data become stale, incomplete, and untrustworthy. That loss of trust means people will view any future efforts on this front with a great deal of suspicion and skepticism, making it much more difficult to implement a reliable solution.

Instead, as quickly as is reasonable, move to an automated method for SCA. Start with the manual method, but only in a limited way to determine the shape and scope of the overall SCA project; at this point try not to roll it out as an official policy and procedure.

Automated Research

Automating the SCA process allows your company to have the most up-to-date view of its software supply chain. It's less error-prone and complete than manual methodologies, but this doesn't come for free. Any sort of automation brings added complexity to the overall software development process.

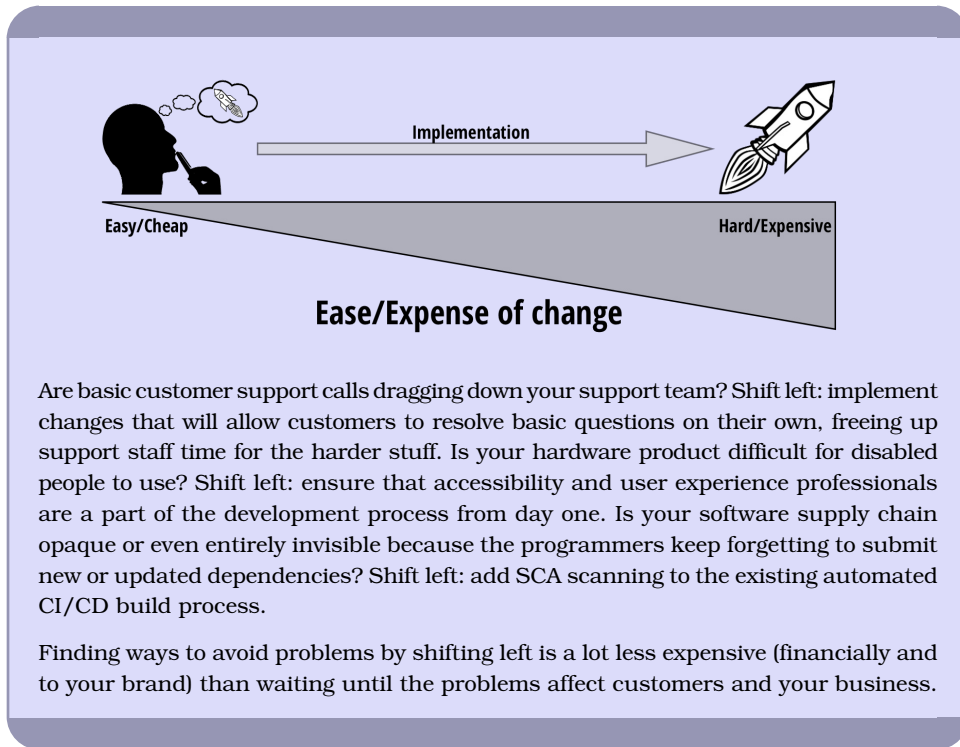
When adding automation, make sure it's documented in such a way that people can easily troubleshoot it should it start to misbehave. Allowing the complex automated system to remain opaque is trading one problem for another. The added complexity is manageable though, and the automation is critical for the ongoing success of your company's SSC awareness and maintenance.

Every software development process will have logical places where automation will fit without adding too much disruption to the workflow of the programmers and other developers. It's likely that your software development team already has some sort of automation in place in the form of continuous integration/continuous deployment (CI/CD). With CI/CD, tests run every time someone makes a change in the source code repository. If the change causes the tests to fail ("breaks the build"), the developers learn about it before the broken code gets delivered to customers. Because these tests run with every source code change, and because the programmers have already taken CI/CD into account in their workflows, this is the perfect place to insert SCA automation. In [Tools, on page 192](#), you'll learn more about some of the tools you can use to help with this automation.

It's Just a Shift to the Left

You're a dastardly evil genius, hell bent on world domination. But first, you need to get that pesky, do-gooder hero out of the way. You devise a complex plan to shoot the hero into space, but you don't leave enough time to test it. The rocket controls overload and melt; the hero escapes. Being a genius, and also evil, you escape as well and return in a year with a new and improved complex plan. This time, you've had your henchmen test everything as they go, so you know your plan will work. To your diabolical delight it does, and you laugh maniacally as you watch the hero starting their one-way journey to outer space. Muahaha.

What our evil genius has done is *shift left*. "Shifting left" is a buzzword that you see all the time lately, largely in the context of DevOps or security, but it can be applied to many different situations. The basic idea is that things get a lot harder and more expensive to fix or change later on in the timeline of a process, so it's better to handle stuff earlier on, to "shift left" on the timeline.



However you approach SCA—manual or automated—make sure that some team or role holds responsibility and is accountable for the process and the data. The role often reports into the Open Source Program Office (OSPO, see [Supply Chain Manager, on page 114](#)). While this role may also include other duties, oversight of the SCA/SSC should be a first-order priority for them. I have never yet seen an SCA/SSC process work if a team or role isn't dedicated to it. Instead, those SCA attempts are half-hearted at best, leading to poor data as well as hard feelings among those whom the attempts impacted. Dedicating staff to the process helps ensure this business-critical information is accurate and accessible.

The Process (Roughly)

A software composition analysis process can become complex but always includes four phases: Scan, Analyze, Remediate, Repeat.

Scan

Scanning is the manual or automated process of learning what external components (dependencies) exist in your software. Sometimes a programming language ecosystem will have build and packaging best practices that can make it easier

to learn about the first-order (direct) dependencies in the software. For instance, software written in the Ruby programming language usually lists its direct dependencies in a file called `Gemfile`. The Java programming language community regularly uses the Apache Maven tool to build the software and manage dependencies.

Packaging tools like these can make it easier to keep track of many of the components used in the software, but they're unlikely to include them all. That's because a lot of software has components written in multiple languages. The back-end code may be written in Scala and the user interface primarily in Django—a web framework written in Python—but with many elements in JavaScript. The site reliability engineers (SREs) and DevOps staff may use OpenTofu, which has its own special configuration language. IT may automate the roll out of new hardware using Ansible, which relies on the YAML configuration language. So while the packaging tools can be helpful, to get a fuller view of the dependencies across your company you'll need your SCA process to look beyond things like Gemfiles and Maven `pom.xml` files.

Supply Webs and Snippets

Many companies don't need to track more than their first-order dependencies—their immediate FOSS supply chain—but all companies will need to start here. If you do nothing else, at the very least create an SCA process that will allow you to scan for and discover this immediate supply chain. The business needs of some companies will require them to take the next step: looking deeper to discover the dependencies of each link in that supply chain. Businesses in highly regulated industries with low risk profiles (recall from [Consider the Risk Profile, on page 164](#)) or that deliver hardware with embedded software may wish to consider taking this step.

In that case, once you feel the SCA scan has a decent grasp on the first-order dependencies in your software supply chain, you can start to climb further into the dependency tree. What dependencies do your first-order dependencies have? What are the dependencies of those *second-order* dependencies? In other words, what is your FOSS supply web? For many companies, knowing those first-order dependencies is enough, but companies in highly regulated industries or with a low tolerance for risk (such as healthcare, finance, or transportation) may need their SCA to go further.

Some companies' SCA goes so far as to look at not only what the dependencies are but also what they contain; these scans include searches for *snippets*, pieces of source code that may match code from software that the company, for its own business reasons, has deemed problematic. For instance, snippet

searching sometimes can locate chunks of source code that were copied from other FOSS projects without attribution or concern for license compliance.

Snippet searching is rarely needed and typically results in so many false positives as to make analyzing the search results an expensive and tedious proposition. Frankly, it's a big headache for all involved. Despite this, some companies find it's worth the considerable trouble and expense.

The Container Complication

It's common now for dependencies to enter the software supply chain by way of containers. A container is a way to package up software along with its dependencies then run it in an isolated way. Containers are, well, self-contained. Everything necessary for running that piece of software is included in a container image (that's called a *container* when you run it; yes, it can get confusing), and that container can run just about anywhere: on your laptop, on a server, or in the cloud.

Containers make it easier to deploy a piece of software, but they're a big pain in the butt where software supply chains are concerned. For one thing, each external container that your team uses to run your software comes with its own dependencies built-in, and sometimes there are hundreds of them. Because of the way some container images are created, it can be difficult to see what all of those included dependencies are. Things like Dockerfiles—which define how to build a Docker container, one of the most popular types—show some dependencies, but even those only go so far.

The acquisition and use of containers as dependencies can go through a different flow than that of a more traditional dependency like a software library. This can cause them to fall through the cracks in scans so they never appear in your view of your software supply chain.

Containers are also designed to be *immutable*. This means that the versions of the software and dependencies that are included in the container image are the only ones that will *ever* exist in the container. If one of the dependencies has a critical security vulnerability, there's no way to update it. Doing so requires a completely new container image—but it also requires knowing that the container is using that vulnerable dependency. Without that knowledge, you're unlikely to recognize the potential danger posed by using a container with an out-of-date dependency.

Finally, it's rare for any container to include the source code for the FOSS software inside it or provide a way to retrieve that code. If that container includes software distributed under a reciprocal Free Software license—and since most containers include software packages from Linux distributions, they most likely do—then the distributor of that container may be violating the terms of the license, unless they're providing some way to access the source code. If you then similarly distribute the container as a part of delivering software, you also must comply with that reciprocal license and ensure source code is available. You'll learn more about FOSS license compliance in [Chapter 12. Maintain FOSS License Compliance, on page 201](#).

So what can you do? As with all matters, a little intention and attention goes a long way here.

- Discover whether your software development teams are using containers from external sources.
- Ask them to track which containers they're using.
- Work with the teams to include a container scanning tool in the automated SSC scans.

Alternatively, it may make sense for your company to build its own containers internally, using known and tracked dependencies, and mandate that only internally built containers may be used in the software. This can be a fair bit of work, but for companies with a low risk profile it can be worth it when compliance audit time rolls around.

Minimum Scanning Results

Your SCA process will need to do more than create a running list of the names of the links in your FOSS supply chain; it also needs to collect metadata about each of those links. What metadata? Well, as with most questions related to the FOSS in your business, the answer is “it depends.”

As you'll see in [Software Bill of Materials \(SBOM\)](#), on page 194, you'll find many different pieces of metadata that you could possibly collect and track for each component. Which ones you choose will depend upon the use to which you're going to put the information. If it's pure recordkeeping and the scan results aren't going to be used for much, or if you're just getting started on your FOSS SCA journey, then you can get away with collecting the minimum amount of data for each component:

- Date and time of scan
- Component name
- Link to component source code
- License(s) of the component
- Attribution for the component

But as your SCA/SSC processes and requirements evolve and become more sophisticated, you may similarly need to evolve your scan results. The compliance team may ask that you track the Software Heritage⁶ link

6. <https://www.softwareheritage.org>

for each component. Security may wish you to associate VEX documents⁷ with certain components. Release Engineering may want to include metadata about which product versions include a component.

The information gathered and stored can vary by company, business, product, and sometimes even team. This can get complicated, but if you start with the minimum information listed here and remain flexible in your SCA processes, they'll be able to evolve as your specific use cases emerge and mature.

No Scan Is 100 Percent Accurate

The complexity of most modern software means that it's possible a scan will never be 100 percent accurate or complete, and that's OK. Don't allow perfect to become enemy of the good enough.

When running a dependency scanning process, be practical. As with all things in software, this is an iterative process; as you learn more, you can change the scan to be more or less inclusive until you're able to get a good picture of the links in your software supply chain. Allow that definition of "good picture" to be driven by business needs. Do your best within the parameters and constraints you're given.

Analyze

Once you have a list of the links in your software supply chain, it's time to learn more about them. As you just learned, this process can be as involved as your business needs demand but should at the very least include a review of which licenses those links use and whether any of those links have any known security vulnerabilities. You'll learn more about what to do with that license information in [Chapter 12, Maintain FOSS License Compliance, on page 201](#). As for the vulnerabilities, you'll find multiple ways to approach them.

The simplest (but by no means simple) approach is to throw money at the problem. An ever-growing list of companies sell services for scanning your SSC. Each of these companies will tout the speed and accuracy of their services and perhaps even their proprietary database of software components. Whether the service they offer is right for your company will, of course, depend on many factors, but one that's often overlooked is software ecosystem: not all vendors provide the same coverage for all software ecosystems. Software written in the Java, Python, or Node.js programming languages may have adequate coverage, but if your company's software uses something like Perl,

7. https://www.ntia.gov/files/ntia/publications/vex_one-page_summary.pdf

Lua, Erlang, or other less-common programming languages, then you should make sure to ask whether the service can support your use case.

The alternative is to do what we in software call *rolling your own*. Software is a part of every facet of our lives now, making it critical infrastructure in fact if not in name. Because software security vulnerabilities can have such a huge impact on our lives, many organizations maintain publicly available databases of them. The most important is the CVE Program from MITRE.⁸ A CVE (Common Vulnerability and Exposure) is a known software vulnerability. MITRE tracks these and assigns them a severity from zero (not especially dangerous) to ten (holy smoke, this is really, really bad). Other public databases include the National Vulnerability Database (NVD) from the National Institute of Standards and Technology (NIST)⁹ and GitHub's Advisory Database.¹⁰ The U.S. Federal Cybersecurity and Infrastructure Security Agency (CISA) maintains a database of software vulnerabilities that are known to have been exploited by evildoers,¹¹

Most of these are based in the United States, but because FOSS is worldwide these databases are available to and critical for people and organizations around the globe. Using these, your software team can develop scans that match your company's software ecosystem and business needs. It'll take time and require maintenance, but it may be a better fit for you.

Pay Attention to Component Versions

Vulnerabilities are important, but they aren't the only thing to be on the lookout for as you analyze the links in your software supply chain. Another important piece of information is the version of the components that your company uses in its software. Out-of-date software components may not appear in vulnerability databases now, but they may tomorrow. Knowing which version of the component your company is using can allow you to react quickly when someone files a CVE against it.

Vulnerabilities aside though, when a piece of software has a new version, it also means that it has new or corrected functionality. This functionality may make things easier for your programmers, but you aren't going to know it even exists if you aren't on the lookout for updated versions of the software. This can be tricky, though, since it requires discovering how and where each piece of software releases and announces new versions. Using and contributing

8. <https://www.cve.org>

9. <https://nvd.nist.gov/vuln>

10. <https://github.com/advisories>

11. <https://www.cisa.gov/known-exploited-vulnerabilities-catalog>

to crowdsourced databases such as ClearlyDefined¹² makes this chore considerably easier, but it's never going to be easy. Some developments, such as the growing prevalence of SBOMs (see [Software Bill of Materials \(SBOM\), on page 194](#)) may help improve this process in the future.

Reduce Scope by Focusing on Load-Bearing Links

If you think this sounds like a lot of work, you're right. It is. The reason I said before that I've never yet seen an SCA/SSC process work if a team or role wasn't dedicated to it is that it's unreasonable to expect people can do this sort of thing well if it's not the focus of their job. Even with a dedicated role or team, though, it's not possible to keep a close eye on every single one of what may be thousands of links in your software supply chain. Thankfully, it's also not necessary.

As you analyze those links, take the time to note those that are of special importance to your business. If your company were a house, these components would be called *load-bearing*: messing with them could bring the entire house crashing down around you. What are the load-bearing links in your software supply chain and which are less critical? For instance, if it's important to your business to be able to process documents uploaded by your customers, the FOSS library that allows for optical character recognition (OCR) of the documents may be load-bearing. The list of load-bearing components will be a fraction of the overall SSC.

This shorter list is easier to manage. Give it the special attention it deserves, ensuring those components are always as up-to-date as reasonable and that the communities and projects that create them remain sustainable (you'll learn more about this in [Chapter 13, Sustain. Maintain. Retain., on page 213](#)). The remainder of the SSC should also get attention but probably won't require as much. In this way you can make it easier to pay attention to those links that are most important to your business needs.

Automate and Document

Whether your company contracts with an external service or rolls its own SCA/SSC solution, ensure that the end result not only can run automatically but also can provide automated reports should it discover any potential problems. You don't want to rely on manual reporting for discovery of critical vulnerabilities, not if you ever want your SCA/SSC team ever to have week-ends, vacations, or sick days.

12. <https://clearlydefined.io>

Also, make sure that the entire automated process has adequate documentation to allow for troubleshooting and training of new staff members. All too often companies defer the task of documentation until it's too late. This inevitably allows small problems to balloon into large ones, causing preventable stress and perhaps even service outage for your customers. Avoid all that trouble; hire technical writers.

Remediate

As you perform an SCA, you'll undoubtedly discover some items that need attention. Perhaps the software is using a component with a reciprocal license, but you haven't yet made source code available. Maybe you've found that the software is using components that are out-of-date or contain security vulnerabilities. Or it may be that your scan discovers that the software teams are using multiple different versions of a single component, and you'd like them all to use the same one if possible. Whatever the issue, you're going to need a process to ensure someone fixes it. This is the remediation step of an SCA process.

At first glance, remediation may look straightforward. “Just” release the source code. “Simply” update the vulnerable component. “Merely” require the developers to use the same version. Quick, easy, boom, done. Unfortunately, it's rarely that simple.

Each of these remediation suggestions may appear easy, but they hide a lot of complexity behind these basic descriptions. Releasing the source code may have business impact. The updated component may have a new API that doesn't work with your existing software, requiring that someone rearchitect the code to make it work. The multiple versions may each have different functionalities required by each team. In each case though, testing and especially time are required to minimize any disruptions to your business and your customers. Since these are important issues, you don't want to lose track of them. You also want to ensure accountability and that someone is responsible for making sure the remediation happens. This is the perfect job for an issue tracker.

Use Issue Trackers

Issue trackers are a normal part of the software development workflow, but they're not only for programmers. Anyone who needs to keep track of tasks over time—especially in coordination with other people—can find value in using an issue tracker, including your SCA/SSC team and OSPO.

If the remediation requires work by a software development team, the issue ticket should go into the issue tracker the team uses so they can include the remediation in their normal workflow. Forcing the people doing the remediation work to look somewhere outside of their workflow to find remediation tickets is a good way to make sure that remediation never happens: out of sight, out of mind.

Aside from providing a tracking mechanism, issue trackers usually include a way to assign a severity to each issue ticket, from low (insignificant) to high (extremely critical). It's a software development best practice not to release software if it has any high-severity issue tickets filed against it. This practice, in combination with your SCA process, can help ensure that the most critical issue remediation occurs before the problem can have too much effect on your business and your customers.

Not every software development team follows this best practice, and even if they do it's still common for people to write up tickets for issues that never get resolved. These issues are important enough to track in a ticket but not important enough for anyone to resolve them immediately. Inevitably, these tickets can build up in a backlog. That backlog is known as *technical debt*. If SCA issues get trapped in the technical debt backlog, that backlog could turn into a minefield, hiding potentially dangerous issues that could explode in your business at any moment.

Your SCA process should empower your SCA/SSC team to raise the alarm of these issues to ensure they receive remediation in a reasonable amount of time. This should include the power to cause these issues to be deemed *blockers*, issue tickets that must be resolved before any new software is delivered to customers. Without these powers, remediation tickets may languish forever in a tech debt backlog.

Avoid Liability Through Remediation

Why is remediation so important? Aside from it being the right thing to do for the sake of software supply chain health and security, it also can save your company a great deal of money and hassle.

For instance, in March 2023, the United States government released its National Cybersecurity Strategy.¹³ In this strategy, companies or organizations that provide software that includes vulnerabilities will be held liable if those vulnerabilities negatively affect those downstream from it (customers, consumers, and others). At this moment those liabilities are undefined, but it's

13. <https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf>

likely that they'll include legal, financial, and perhaps even criminal charges against the company at fault.

By engaging in an SCA process, growing your awareness of your software supply chain, and ensuring remediation of any issues, you're performing the due diligence required to ensure the safety of those who use software created by your company. Neglecting the remediation equates to knowing that you have a problem but refusing (passively or otherwise) to do anything about it. This is worse than not knowing at all and certainly makes the company complicit in any damage that lack of remediation may induce.

Repeat

The final step in your software composition analysis process is to repeat, repeat, repeat. The initial SCA output provides a snapshot of your software supply chain at a moment in time, but your own software is constantly evolving, as are all of those FOSS components on which it depends. If you don't regularly repeat that SCA process, then you're going to miss a lot of changes in your SSC.

How frequently should you repeat the process? In a perfect world, all companies would have an automated SCA process—including notification and remediation—that runs more or less constantly. If your company is in a highly regulated industry or otherwise has a low risk profile, it should certainly aim for this ideal. But most of us don't live in a perfect world. For whatever reason, your company may not be able to run the process that frequently. This can be OK. Do the best you can with what you have.

At the very least, make it a priority to run your SCA process once a year. This is the bare minimum, and for many companies it may fall far short of their business needs, but it's literally better than nothing. Even an SCA performed as infrequently as this can still help you identify problems before they become crises, or spot crises that you didn't know were there. If nothing else, repeating the SCA yearly provides a fresh point-in-time snapshot of your software supply chain, so if your security team notifies you of an exploited critical vulnerability identified in the world, you'll have the data at hand to determine whether that vulnerable component is a part of your supply chain.

Raiders of the Last Archive

If you recall from [Inbound FOSS Integration Policies: Vendoring, on page 169](#), some companies add an extra step to their SCA process: archiving every component and container used in their software development process. When building the software, these companies will perform the build using components from these *artifact stores* or *repositories* rather than pulling the components from the internet. It's an awful lot of work to create and maintain these artifact stores, so why do these companies do it?

Building software using a local cache of components can speed the build process considerably. It also allows these companies to build their software even if the components aren't available on the internet. While the usual cause for this is network or cloud-services related, sometimes the reason is more nefarious or activist in nature.

It's rare still, but recent years have seen several instances of maintainers of critical components in the FOSS supply chain intentionally sabotaging their own code and then releasing it. This most commonly happens in the npm software ecosystem but has been seen elsewhere as well. These maintainers each have their own reasons for their actions, but the results are always the same: software built by downloading the latest version of components from the internet will break.

This causes about as much havoc as you'd expect, and for many companies the risk of downtime caused by a begrudged maintainer is more than they can afford. Building their software from locally maintained artifact stores ensures that they're never caught by surprise by a malicious or activist component update.

Tools

Your company can do OK by using spreadsheets to track its software supply chain, but they'll only get you so far. At some point in your SCA experience, you'll need to upgrade to dedicated tools to help with each step of the process.

Many commercial options exist for each step of the SCA process, but for better or worse the picture here is constantly shifting. The SCA startup landscape was already crowded before 2021, but EO 14028 created a bit of an SCA gold rush. The number of commercial tools offering SCA functionality has exploded. Actually, wait. I shouldn't use past tense. The number of commercial SCA tools *is* exploding and is likely to continue doing so for a few years yet, until the inevitable consolidation occurs by way of mergers and acquisitions (M&A) as the apex predators appear in this evolving ecosystem. Because this ecosystem is evolving so quickly, I won't make recommendations for commercial SCA tools. Such recommendations would be out-of-date by the time you read this book.

The situation is more stable on the free and open source software side. While FOSS SCA tools continue to evolve and improve, unlike commercial tools they're not popping up everywhere like mushrooms after the rain. This is because most of the FOSS tools started years ago to aid companies in tracking whether they're in compliance with the FOSS licenses in their software supply chains (see [Chapter 12, Maintain FOSS License Compliance, on page 201](#)). As these tools have been around for a while and are good at what they do, most people wisely decide to contribute to them rather than reinvent the wheel by creating their own new FOSS tools.

Here are a few of the most popular and useful FOSS SCA-related tools available today:

- **package-url (purl):**¹⁴ This is less a tool itself than a building block for both FOSS and commercial SCA tools. Using purl, a tool can uniquely identify a specific software package, including information like version, operating system, computer architecture, and more. This unique identifier is critical for ensuring that all SCA process components are using the same language when communicating about the components in your software supply chain. As a FOSS building block, your company can use it directly if rolling your own SCA solution.
- **ScanCode:**¹⁵ This is another building block that's critical to many FOSS and commercial SCA tools. Both it and purl originate from nexB, a FOSS-friendly company that's been doing SCA longer than just about anyone else. ScanCode has become the de facto standard tool for performing scans to uncover details about a software supply chain.
- **FOSSology:**¹⁶ A full-featured SCA tool, FOSSology has been under active development for more than a decade and a half. It provides both a multi-tenant web interface and a command-line interface for its multiple functions.
- **Open Source Review Toolkit (ORT):**¹⁷ ORT is a powerful suite of SCA-related components. These components are designed to be used from the command line, but they also come with CI/CD support built-in, making it easier to include them in your software build pipeline. One of the unique features of ORT is that it allows you to enforce the company's inbound FOSS policy automatically with its Policy-as-Code functionality.

14. <https://www.aboutcode.org/projects/package-url.html>

15. <https://github.com/nexB/scancode-toolkit>

16. <https://www.fossology.org>

17. <https://oss-review-toolkit.org/>

- Tern:¹⁸ If you want to see what's in those containers your software development team is running, then Tern is the tool for you. It will create a complete picture of the components of the container's software contents and then export it to standardized formats.
- ClearlyDefined:¹⁹ After nearly forty years, you'd think that the FOSS community would have a standard method for defining the license under which they're releasing software. That's not the case, alas, but ClearlyDefined is here to help. It's an open and growing database and service providing curated license data for FOSS components.

Any of these tools would be a good addition to the SCA processes of most companies. Because they're FOSS, and therefore not in competition, people get the added benefit that many of these tools are designed to work well together. The ORT project has made this a particular focus, ensuring it integrates smoothly with other tools using standard formats and interfaces. Most of them also perform valuable roles in generating a Software Bill of Materials, also known as an SBOM. You'll learn about that next.

Software Bill of Materials (SBOM)

The rising awareness and use of software bills of materials may help to make your SCA process more streamlined and complete. According to both CISA and NTIA,²⁰ a software bill of materials (SBOM) is defined as "...a nested inventory for software, a list of ingredients that make up software components." When you receive a physical delivery of some sort, the bill of materials tells you what's inside the box. Similarly, when you use software created outside of your organization, the SBOM tells you what's inside that software. The SBOM is a file that declares the SSC for that specific piece of software.

The concept of an SBOM has been around for many years now, but it's finally gaining mindshare and adoption as companies start to become aware of their software supply chains. This growth in mindshare and adoption got a big boost when U.S. President Joe Biden released Executive Order (EO) 14028, which among other things mandates that vendors providing software to the United States government must start providing SBOMs as well as the software itself.

While officially the scope of the EO is only companies working with the U.S. government, in practice the EO was like a large rock thrown into a still pond.

18. <https://github.com/tern-tools/tern>

19. <http://clearlydefined.io/about>

20. <https://www.cisa.gov/sbom>

The ripple effects have inspired software providers around the world to start looking at their own software supply chains and investigate both receiving and creating SBOMs themselves. The NTIA, NIST, and CISA, despite being U.S. government agencies, have been hosting and fostering SBOM discussions with worldwide stakeholders. The SBOM guidance provided by these agencies therefore is largely applicable to software users and providers in any country on Earth.

What To Include in an SBOM

An SBOM can contain a lot of information about each and every component in a piece of software's supply chain, but the currently accepted minimum data for each component is a fairly short list:²¹

- Component name
- Component supplier name
- Component version
- Other unique identifiers for the component: Software Identification Tags (SWID), Package Uniform Resource Locators (PURL), and so on
- Component relationship to the software (dependency of...)
- Author of the SBOM
- Timestamp of SBOM creation

SBOM Formats

A provider can supply an SBOM in any number of different formats, including a file of comma-separated values (CSV) or an Excel spreadsheet. But while they *can* provide it in any format, two formats are preferred and are best practice: SPDX and CycloneDX. These appear to be winning the popularity race at the moment, but SBOMs and SSC awareness are rapidly evolving concepts, so perhaps some other format may take the lead at some point in the future.

SPDX Example

SPDX is an open international standard (ISO/IEC 5962:2021) actively developed under the Linux Foundation. It defines a format for SBOM documents that are readable by humans and computers alike. Initially focused on gathering and communicating data necessary for copyright and license

21. https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

compliance, over the decade-plus of its existence SPDX has grown to include all necessary data for a software supply chain. Here's an excerpt from an example SPDX document (licensed CC0):²²

```
{
  "SPDXID" : "SPDXRef-DOCUMENT",
  "spdxVersion" : "SPDX-2.3",
  "creationInfo" : {
    "created" : "2022-10-23T15:44:16Z",
    "creators" : [ "Person: Gary O'Neill", "Tool: spdx-maven-plugin" ],
    "licenseListVersion" : "3.18"
  },
  "name" : "examplmaven",
  "dataLicense" : "CC0-1.0",
  "documentDescribes" : [ "SPDXRef-example" ],
  "documentNamespace" : "http://spdx.org/documents/examplmaven-0.0.1",
  "packages" : [ {
    "SPDXID" : "SPDXRef-junit",
    "copyrightText" : "UNSPECIFIED",
    "description" : "JUnit is a regression testing framework written by
    Erich Gamma and Kent Beck. It is used by the developer who implements
    unit tests in Java.",
    "downloadLocation" : "NOASSERTION",
    "filesAnalyzed" : false,
    "homepage" : "http://junit.org",
    "licenseConcluded" : "NOASSERTION",
    "licenseDeclared" : "CPL-1.0",
    "name" : "JUnit",
    "originator" : "Organization: JUnit",
    "summary" : "JUnit is a regression testing framework written by
    Erich Gamma and Kent Beck. It is used by the developer who
    implements unit tests in Java.",
    "versionInfo" : "3.8.1"
  }, {
    "SPDXID" : "SPDXRef-log4jslf4jbinding",
    "copyrightText" : "UNSPECIFIED",
    "description" : "The Apache Log4j SLF4J API binding to Log4j 2 Core",
    "downloadLocation" : "NOASSERTION",
    "filesAnalyzed" : false,
    "licenseConcluded" : "NOASSERTION",
    "licenseDeclared" : "NOASSERTION",
    "name" : "Apache Log4j SLF4J Binding",
    "summary" : "The Apache Log4j SLF4J API binding to Log4j 2 Core"
  }
],
```

22. <https://github.com/spdx/spdx-examples/blob/master/example8/spdx/examplmaven-0.0.1.spdx.json>

CycloneDX Example

CycloneDX is an open BOM standard developed and maintained by the Open Worldwide Application Security Project (OWASP) Foundation. Like SPDX, it's readable by humans and computers. Unlike SPDX, CycloneDX provides BOM formats for things like hardware and operations as well as software. Following is an excerpt from an example of a CycloneDX SBOM (also licensed CC0):²³

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.2",
  "serialNumber": "urn:uuid:699b6458-60da-4f52-b1b3-34915dc01eb6",
  "version": 1,
  "metadata": {
    "timestamp": "2020-08-03T01:28:52.765Z",
    "tools": [
      {
        "vendor": "CycloneDX",
        "name": "Node.js module",
        "version": "2.0.0"
      }
    ],
    "component": {
      "type": "library",
      "bom-ref": "pkg:npm/lhc-vdm-editor@0.0.1",
      "name": "lhc-vdm-editor",
      "version": "0.0.1",
      "description": "A code and file editor for VdM scans.",
      "licenses": [
        {
          "license": {
            "id": "Apache-2.0"
          }
        }
      ],
      "purl": "pkg:npm/lhc-vdm-editor@0.0.1",
      "externalReferences": [
        {
          "type": "website",
          "url": "https://github.com/CERN/lhc-vdm-editor#readme"
        },
        {
          "type": "issue-tracker",
          "url": "https://github.com/CERN/lhc-vdm-editor/issues"
        }
      ]
    }
  }
}
```

23. <https://github.com/CycloneDX/bom-examples/blob/master/SBOM/cern-lhc-vdm-editor-e564943/bom.json>

```

    {
      "type": "vcs",
      "url": "git+https://github.com/CERN/lhc-vdm-editor.git"
    }
  ]
},

```

Most of the FOSS and commercial tools for SCA will export your SSC to create an SBOM document formatted as SPDX, CycloneDX, or both. The communities who maintain each of the formats also provide software libraries to integrate SBOM creation functionality into internally developed SCA tools and processes. All of these options make it considerably easier to generate and share SBOMs in standardized formats that other organizations and processes can ingest and understand.

Using SBOMs Isn't Easy (Yet)

But here we hit a bit of a snag: while a fair bit of support is out there for *generating* SBOMs in a standard format, the ecosystem is playing catch-up on supporting *ingestion* or using of SBOMs. This means that even if your company's software vendors do provide you with SBOMs (for which you may need to ask...sometimes repeatedly), you may not be able to do much with those documents beyond storing them in an online shared drive in case you need to search them manually. Of course, I write that today, but this is yet another situation where the ecosystem is evolving so quickly that it may not be true tomorrow. Eventually, it'll be possible to ingest externally provided SBOMs into your SCA tooling and generate reports and alerts based upon them.

While SBOMs are a promising development in the software supply chain world, they're not a magic wand that you can wave and make everything better and easy. Relatively few commercial software vendors provide SBOMs at this point, even if requested. SBOMs are nearly unheard of in the FOSS world, so at the moment almost no open source components will come with an SBOM document. The tooling concerns I just mentioned don't help the situation; formatting and interoperability are also problems.

Despite the existence of two well-developed and well-considered SBOM formatting standards, it's still common for many of the few SBOMs you'll find to be either in a proprietary format or to be formatted as CSV or Excel. This picture is changing, but it hasn't changed *yet*. In the meantime, those who wish to incorporate SBOMs into their SCA and SSC processes must deal with the growing pains of an evolving ecosystem.

As you can see, learning about and maintaining a software supply chain can be complicated. It's always worth the effort though, if only so your business and customers aren't rudely surprised by a broken link in that chain. That SSC awareness is useful for more than looking for broken links. It also can keep your company out of legal trouble. How? By learning how to stay in compliance with licenses and regulations.

Maintain FOSS License Compliance

In [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#), you learned about your software supply chain (SSC). One of the many benefits of getting up close and personal with your SSC is that it allows you to ensure that you're complying with all applicable licenses and regulations. This is impossible without knowing what links are in your software supply chain, since obviously you can't comply with licenses that you don't know you've agreed to.

What Is License Compliance?

What is license compliance, and why should you care about it? If you recall from [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#), a copyright holder uses a license to say *who* can use their work and *how* they're allowed to use it. License compliance is the process of ensuring that your company has the right to use a copyrighted work (are one of the *who*) and that your company is following the conditions of the licenses (the *how*).

Remember, in the case of free and open source software, using the software means you agree to the terms of the licenses under which the copyright holders of the software have released it. If your company is using the FOSS, it has agreed to the terms of the license, even if it doesn't realize that the FOSS component is in their SSC. The license compliance process is the inspection of every link in your software supply chain to ensure the company does what it promised to do (*complies*) by agreeing to the terms of the licenses of those links. If the company does not comply with all of the terms and conditions of a license, it's infringing on the license and at risk of legal action.

What Qualifies as Infringement?

It's easy to forget with what looks like the free and easy “anything goes” environment of FOSS, but licenses are legal documents. Failure to follow all of the terms and conditions of a license is a violation of that license and therefore infringement of the copyright, and the perpetrator can face legal action.

For example, I have a tattoo of artwork that I commissioned from a skilled artist. During the process of commissioning the work, we negotiated that she retains copyright over it, but she granted me an exclusive license to use it for a tattoo and to print the art for personal use. Now if I were to post the art on my website, or use it in this book, I would violate the license, infringe on the artist's copyright, and she would have some very stern words and perhaps a court summons for me.

The terms vary across all FOSS licenses, so it's important to be aware of each of the licenses represented in your company's software supply chain and their terms. Ignorance is no defense, especially not for a company that has plenty to lose. This is why it's important that your company always have someone—a lawyer or a knowledgeable layperson in your OSPO—read and understand the licenses in your SSC.

License Compliance Enforcement

As with all matters concerning intellectual property, license infringement and enforcement are complex and have many sharp edge cases. The simplest explanation of this complexity is that any copyright holder has the right to enforce a license for their part of a copyrighted work. Some FOSS components can have dozens or even hundreds of copyright holders, and each one of them can be on the lookout for companies and individuals who aren't complying with the terms of the license of the FOSS component. Furthermore, they're within their rights to take legal action.

Don't take this as incitement to paranoia, remaining constantly on edge awaiting notification that your company is being sued for copyright infringement because it didn't comply with all the terms of a FOSS license. In fact, it's rare that matters involving FOSS license compliance end up in a court of law. Part of the reason for this is that it's relatively easy to comply with FOSS licenses once you realize your company is party to them.

The other reason is that the overwhelming majority of FOSS contributors don't want to go to court; they want people to comply with the terms of the

license. Because of that, it's best practice in the FOSS world, when a copyright holder discovers a case of license infringement, to notify the company of the problem privately and ask them to come into compliance.

The FOSS world understands that mistakes happen and prefers to assist in correcting the mistake rather than assigning blame and punishment through legal action. This approach, captured in *the principles of community-oriented enforcement*, was created in 2015 and is practiced by key FOSS nonprofit organizations such as Software Freedom Conservancy.¹ These principles play a valuable role in maintaining the culture and ecosystem of free and open source software by reducing the paranoia around license compliance, reinforcing the importance of the underlying philosophies and intentions of FOSS, and providing accessible and knowledgeable guidance for FOSS copyright holders.

Beware the Trolls and Profiteers

Despite this, once in a while copyright trolls and profiteers appear in the FOSS world. These take the form either of companies that use FOSS components but willfully neglect to come into compliance with the licenses—even when notified of infringement—or of individual copyright holders who prefer to jump directly to legal action in an attempt to extort payment from companies that unknowingly infringed on the holder's copyright by violating the terms of the FOSS license. Thankfully these profiteers are extremely rare and usually face scorn from the FOSS community when they do arise. This community judgment helps to keep the profiteers at bay.

Even though the negative and legal ramifications for accidentally infringing on a FOSS license are vanishingly small, it's common to see some folks spreading fear, uncertainty, and doubt (FUD) about it in the media. This fearmongering often originates from commercial SCA companies as a part of their marketing strategy, but not all of it. A large amount of wider-spread FUD is centered around reciprocal (copyleft) licenses. Prior to believing any type of negative FOSS license messaging you may see, I strongly encourage you to press the paranoia pause button and recognize the messaging for what it is: manipulation for largely commercial ends.

Don't worry and don't believe the FUD. If your company is aware of the links in its software supply chain, does its good-faith best to remain in compliance with the licenses represented in it, and fixes compliance problems if they arise, you'll be fine.

1. <https://sfconservancy.org/copyleft-compliance/principles.html>

Distribution: It's Complicated

If you recall from [Distribution, on page 60](#), distribution of software is the action that triggers the obligations in FOSS licenses. If you're using the FOSS components internally, and the software you're creating has no chance of ever leaving the company in any way, then according to the FOSS licenses you're not really required to do much of anything.

However, if someone outside of your organization legally ends up with your company's software in their possession, including if it's embedded in a piece of hardware, then the software was distributed and your company must comply with the obligations of any FOSS licenses. And remember: the obligations of certain FOSS licenses have definitions of distribution that include making the software available to others as a service over an internet network (software as a service [SaaS]).

Sure, you could special-case the situation where the software being developed is intended for internal use only, but why add that level of complexity to an already difficult SCA process? If you treat all FOSS components in the same way, whether they're used internally or for external use, you'll be prepared should that internal software be distributed. After all, you never know when upper management is going to decide that a staff-only utility would make a great product and ask you to ship it by next quarter.

Consequences for License Infringement

As you learned previously, thanks to the principles of community-oriented enforcement, for FOSS it's usually enough to come into compliance once someone points out infringement to you. Sometimes, though, a company or individual either can't or won't come into compliance for whatever reason. What happens in a case like that? As you've probably come to expect, the answer to that is "it depends."

Infringing on license terms can mean you can lose your license to use the software in question. For instance, the GPL-3.0 license includes a termination clause:

8 Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

Legal penalties are also a very real possibility. These will vary by jurisdiction, the type and scope of the infringement, and many other variables. The penalties themselves can be expensive, but also consider possible legal fees. Some penalties may even include paying the legal fees of the plaintiff in the infringement case.

The risks of violating software licenses are not theoretical and not limited to FOSS licenses. For instance, Microsoft has a long history of software license auditing and compliance verification, leading to hefty penalties for companies found using unlicensed copies of Microsoft software products.²

In the FOSS world, we have the example of minIO—an open source object storage platform—terminating the license used by the Nutanix company due to noncompliance.³ Also, in the well-known BusyBox cases, copyright holders sued several embedded systems providers for violating the terms of the project's FOSS license.⁴

More recently, in 2021 the Software Freedom Conservancy made history for initiating a right-to-repair lawsuit against Vizio, stating that Vizio's violation of the GNU Public License (GPL) also violates the rights of the consumers who stand to benefit from the terms of the license.⁵ This is the first legal case from the perspective of the third-party beneficiaries of FOSS licenses, so it's being watched keenly by legal experts and FOSS community members all over the world.

Never forget that while the risks are real, they're also easily preventable. Remaining aware of the links in your software supply chain and the licenses under which they're released is critical for maintaining compliance and sidestepping any potential problems.

2. <https://web.archive.org/web/20221201124614/https://www.techrepublic.com/article/four-microsoft-licensing-missteps-that-could-cost-you-big-bucks/>
3. <https://blog.min.io/nutanix-objects-violates-minios-open-source-license/>
4. https://en.wikipedia.org/wiki/BusyBox#GPL_lawsuits
5. <https://sfconservancy.org/copyleft-compliance/vizio.html>

Basic License Compliance

Yes, obligations vary across the FOSS licensing landscape. That doesn't mean license compliance needs to be a convoluted mess. In fact, three basic steps will cover the majority of your license compliance needs (and one of them is optional for a subset of licenses): checking for compatibility, giving attribution, and providing source code.

Check for Compatibility

FOSS licenses generally mingle pleasantly together in a software codebase, but a few don't get on well at all. For instance, Apache-2.0 and GPL-2.0 don't play nicely together,⁶ which can be problematic since these two FOSS licenses are popular. When reviewing the licenses represented in your SSC, make sure they're compatible with each other. The license compatibility matrix in [FOSS License Compatibility, on page 61](#), will help here, but you may also need legal assistance for anything more than the most basic case.

Should you locate potential incompatibilities, see how the relevant components are used in the software. They may not be used in a way that triggers the licenses' terms that lead to incompatibilities. If those terms *are* triggered, there's nothing for it but to replace one of the components, thus eliminating the incompatibility. This, as you can imagine, can be a pain. As you'll learn later in this chapter, automated policies can help to avoid this particular problem.

Give Attribution

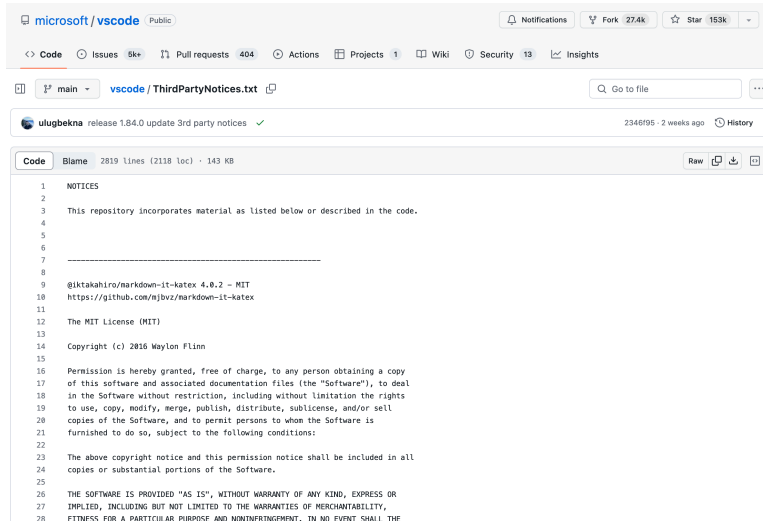
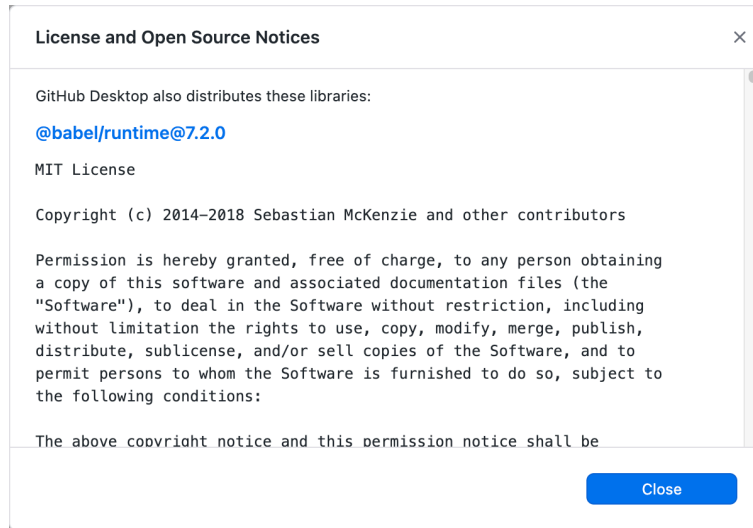
When you benefit from the work of someone else, it's polite to give them credit (attribution) for their contributions. In FOSS, it's not only polite but also required by the terms of most licenses, both permissive and reciprocal. For instance, the highly permissive MIT license has but one license term, and it includes attribution for the copyright holder:

The above copyright notice and this permission notice (including the next paragraph) shall be included in all copies or substantial portions of the Software.

We usually call these attributions *notices* or *NOTICE files*, but you'll find them under a variety of names and also in a variety of formats. These notices should include the information required by the license (usually, as shown in the preceding MIT example, the copyright notice and the text of the license) and must be posted somewhere users of the software can access it.

6. <https://www.apache.org/licenses/GPL-compatibility.html>

For example, as shown in the following figures, the open source GitHub Desktop Git version control client includes the notices in a link that someone can reach via the About dialog box for the software, while Microsoft's open source VS Code editor lists the notices in a file named ThirdPartyNotices.txt on its public version control source code repository.⁷



7. <https://github.com/microsoft/vscode/blob/main/ThirdPartyNotices.txt>

Most SCA software can provide this information, and some (such as Open Source Review Toolkit (ORT)) can generate the notice files for you.

In [Software Bill of Materials \(SBOM\), on page 194](#), you learned about SBOMs. If your company either generates or receives SBOMs, then you're in luck! A correctly formatted SBOM will include all of the information required to generate notice files, making the process considerably easier.

Provide Source Code

If the software your company creates includes reciprocally licensed components, you're required to provide the source code for the derivative works created using those components. This is the “reciprocal” nature of these FOSS licenses: you've benefitted from this FOSS project, so others should be able to benefit from your own work that's based on it.

What constitutes a “derivative work” is a complex matter, as you learned in [Derivatives, on page 58](#). I feel it's safest to interpret it as broadly as reasonable rather than attempting to walk the thin line of “derivation or not.” As in so many cases where licenses are concerned, you may need to bring in the lawyers to sort it all out.

Regardless, once someone has decided that the software is a derivative of a reciprocally licensed FOSS component, you'll need to ensure that people can access the source code for that derivative work. Out of any number of ways to do this, the easiest (especially in the case of software embedded in hardware) may be to provide a link to the source code as well as the instructions and support utilities necessary for building or compiling the code to create a functional version of the derivative work.

Ignorance Is No Excuse

Regardless of what you read here or elsewhere, *always read and understand the licenses in your software supply chain*. Remember: by using the software you're agreeing to the terms of those licenses. Would your company sign a contract without reading it first? Probably not, but that's similar to what happens when people use FOSS components without being familiar with the terms in their licenses.

Always have someone—a lawyer or knowledgeable layperson—read and understand the licenses in your SSC. Also, make sure they verify that those licenses are actually open source—that is, reviewed and approved by the Open Source Initiative. A lot of licenses out there claim to be open source but actually include terms that violate the standard Open Source Definition.

These *fauxpen source* licenses can cause unexpected and unpleasant surprises for your company, its business, and its customers.

The Container Complication Redux

You learned in [The Container Complication, on page 184](#), containers hold all the software needed to run a program, they'll run just about anywhere, and they're a big pain in the butt when you're trying to get insight into your software supply chain. Therefore it shouldn't surprise you to learn that this also makes containers a big pain in the butt where license compliance is concerned.

Many of the containers used in software development are created externally and brought into your company by way of a *container registry*, which is a sort of search and storage engine for container images. Because the container is created elsewhere, it means that someone somewhere has distributed that container. You know what distribution does, right? Right: it triggers FOSS license terms. Many container creators and distributors aren't aware of this (but, again, ignorance is no excuse), so by the time the container reaches your software development process, it may already be out of compliance with the licenses of the software it's distributing.

Now, if you're only going to be using the container for software that's used internally at your company, you probably don't have much to worry about here. After all, you weren't the one who distributed the container and its software, so you're not the one out of compliance with the licenses. But if there's any chance at all that the software will be available outside of your company? Well, you got trouble, my friend. With a capital T, that rhymes with C, and that stands for Containers.

It's OK, this is fixable. The Tern open source tool^a allows you to have a look inside of a container and can create an SBOM of the contents. If your inspection of that SBOM reveals any potential FOSS licensing problems, you can either ask the container creator to come into compliance or re-create the container internally in a way that complies with all relevant FOSS licenses.

a. <https://github.com/tern-tools/tern>

Inbound License Policies

The Open Source Initiative (OSI) has verified a little over 100 licenses as adhering to the Open Source Definition.⁸ This means that only a limited number of FOSS licenses are out there, are you're unlikely to come across the majority of them. Furthermore, as you learned in [Types of FOSS Licenses, on page 44](#), FOSS licenses are of only three types: permissive, strong reciprocal (copyleft), and weak reciprocal.

8. <https://opensource.org/licenses/>

The limited number and types of FOSS licenses can make it easier to create your initial set of policies around how to handle FOSS licenses. These are the commodity policies: they apply to large sets of licenses and can handle about 80 percent of the license compliance questions that will come up in your software development process. The following are some example commodity policies:

- Only use components with OSI-approved licenses.
- When using permissively licensed FOSS components, log the copyright statements, licenses, and location of the source code.
- When using reciprocally licensed FOSS components, log the copyright statements, licenses, and download and archive the source code and supporting build scripts and documentation in the artifact repository.
- Components under any other license are exceptions. For these, file a ticket for review prior to integrating the component into the software being developed.

The actual commodity policies for your company will undoubtedly be different, since you'll have particular business needs not represented here. However, you'll always have to handle that exceptional group of components, so you'll always need custom policies as well. Some of those exceptions may be relatively easy to manage.

For instance, many FOSS projects have a license that's some sort of variant of the MIT License. The MIT License is approved by OSI, but these variants are not and so are exceptions. On review, you may find that the changes in a particular MIT variant are minimal or otherwise present no risk to your business or company. The custom policy for that variant would be as easy as categorizing it as a permissive license and treating the component according to that commodity policy.

An example of this is the Tcl/Tk license. It isn't approved by OSI (possibly because the license stewards never submitted the license for approval) and therefore would normally be disallowed by the policy "Only use components with OSI-approved licenses." Have a look at the substantive portion of the Tcl/Tk license:

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

Compare that with the MIT License:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

You see the Tcl/Tk license is remarkably similar to the highly permissive MIT License. It's typically considered safe to create an exception for Tcl/Tk to treat it in the same manner as MIT. This can be one of the custom policies your company establishes for its inbound FOSS.

Don't Forget Your Risk Profile

As you're developing your inbound FOSS licensing policies, keep in mind the risk profile of your company and its business, including the regulatory environment in which it operates. Highly regulated industries may face additional requirements that require compliance beyond licenses. Companies with software-based business models may wish to consider how to manage the product such that releasing source code to comply with reciprocal licenses can become a strategic strength. You'll learn more about this in [Chapter 19, The Strategy of Releasing FOSS, on page 297](#).

All organizations should keep mergers and acquisitions (M&A) in mind. M&As nearly always involve some sort of software audit, including license compliance. Having inbound license policies in place and in action already can ease that part of a complex process.

Whatever inbound FOSS license policies your business needs, you'll find those policies to be invaluable. Clear policies and education about them empower software developers and others to do the right thing where FOSS licensing is concerned and to know why those policies exist in the first place, reinforcing the importance of following the policies. Revisit [Chapter 9, Establish Effective Inbound Policies and Procedures, on page 145](#), for more information on this.

Don't Legislate. Automate.

As you learned in [Enforce the P&P, on page 156](#), having policies is one thing, but enforcing them is another one entirely. I assure you, your SCA team doesn't want to manually inspect every single inbound FOSS component to

ensure that the company is staying in compliance with the terms of the license. As little as your SCA team wants to do it, your software development team wants to do it even less. Rather than risk an uprising by forcing skilled staff members to perform tedious tasks, keep the peace through automation.

The commodity policies that you created are perfect candidates for automation. Some SCA tools, such as Open Source Review Toolkit (ORT),⁹ have built-in functionality for automating enforcement of inbound FOSS license compliance policies. If your company rolled its own SCA tools and pipeline, it may need to build an inbound policy enforcement module that runs during the CI/CD or build cycle of the software development workflow.

When automating the policies, you may discover that it's difficult to express the rules for a policy in a way that enables automation. Should you come across a policy like this, it's a sure sign that the policy is unnecessarily complex for humans to understand and follow, let alone an automation engine. The policy likely is a victim of what's known in software development as *premature optimization* and suffering from a bad case of YAGNI (You Ain't Gonna Need It).

It's tempting when creating a policy to try to consider all possible futures and have cases in the policy to optimize for every possible situation, when the truth is that those futures and situations never arise. The policy has been prematurely optimized for fantastical eventualities and gratuitously encumbered by concomitant complexities (much like this sentence).

Take a good, hard look at the policy and reduce it to its core, stripping away the YAGNI elements that are making it difficult to automate and to understand. Once you've used the policy for a while, you'll see whether it needs more refinement. That's the time to optimize it, not before.

Complying with the terms of the licenses of the links in your software supply chain is the right thing to do, and it also protects your company from preventable risks. However, you'll also need to consider other less obvious risks for your inbound FOSS supply chain. What if a critical, load-bearing link in that chain were to break? What happens to your business should a FOSS project die or disappear suddenly? Like the risks of license compliance, this one is also preventable and requires understanding and maintaining the sustainability of links in your FOSS software supply chain. You'll learn about this in the next chapter.

9. <https://oss-review-toolkit.org/>

Sustain. Maintain. Retain.

It's vital to your business that you accept your responsibility for understanding the sustainability of the critical links in your FOSS supply chain. Any chain is only as strong as its weakest link. Do you know the strength of the links in your free and open source software supply chain? Don't hope or assume that those links are and will remain strong; take action to ensure their health and longevity.

Sustainability Is a Business Matter

Typically when you hear the word “sustainability” it's in the context of the planetary environment: reduce, reuse, recycle, renewable energy. This is only one piece of the overall sustainability effort, which directly impacts the well-being of your company and business. Others include the wellness of your employees and the society in which they live and the availability and longevity of the resources necessary for your company to do business, including free and open source software.

As you'll learn, the concern for sustainability in a corporate context is far from new, yet many organizations still struggle with how to get started. The first step on this path is to learn about the existing efforts on this front.

Brundtland Report

In the mid-1980s the United Nations brought together scientists, business leaders, and politicians to collaboratively take a good, hard look at the state and future of the world. In 1987 this commission published *Our Common Future: The World Commission on Environment and Development*, which is better known as *The Brundtland Report*.¹ The report—which, again, was

1. https://en.wikipedia.org/wiki/Our_Common_Future

largely written by businesspeople and politicians—concludes that contemporary corporate and political operating procedures were *not sustainable* and, if things continued along their current path, would end very poorly for the human race.

The Brundtland Report defined sustainability as “development that meets the needs of the present without compromising the ability of future generations to meet their own needs.” It recognized the need for balance between economy and ecology rather than the pursuit of economy at the expense of all else. Among other things, the report identified three significant and interconnected crises:

1. Poverty reduction
2. Wealth redistribution
3. Gender equality

The authors of the report found that these were crises because without attention, these three issues would be the primary drivers of the decline of the planet and our species (read: business resources and customers). The crises are interconnected because the authors discovered that it’s not enough to focus on a single one, or even on one at a time in sequence. To prevent the decline, all three crises must receive attention in parallel. A company simply hiring more women would not be adequate; it must also ensure those women are paid and treated the same as men as well as ensure that all its employees and those of its suppliers are paid and treated fairly for their labor.

In 2015 the United Nations released its 17 Sustainable Development Goals (SDG),² the spiritual successor to *The Brundtland Report*. The SDGs build upon and extend the findings of the previous report, providing additional details to make them more actionable. These details include metrics to help gauge whether the goals are on track.

The Brundtland Report (now aided by the SDGs) led directly to the corporate sustainability movement. Wikipedia defines *corporate sustainability* as “an approach aiming to create long-term stakeholder value through the implementation of a business strategy that focuses on the ethical, social, environmental, cultural, and economic dimensions of doing business.”³ Corporate sustainability plans have become an integral piece of business strategies for most large corporations and, according to research from MIT

2. <https://www.un.org/sustainabledevelopment/>

3. https://en.wikipedia.org/wiki/Corporate_sustainability

Sloan,⁴ provide measurable business benefits, such as increased investment, reduced turnover, and advanced innovation.

It's All About Ecosystems

Any discussion of sustainability, no matter what's being sustained, must pivot around the concept of ecosystems. The online Merriam-Webster dictionary has two ecosystem definitions that are relevant here: "the complex of a community of organisms and its environment functioning as an ecological unit" and also "something (such as a network of businesses) considered to resemble an ecological ecosystem especially because of its complex interdependent parts."⁵

The matters of complexity and interdependency are key for sustainability: impacts on one part of an ecosystem can have impacts on all of the others, even elements you didn't expect. In the natural world, the removal of an element of an ecosystem—a predator, for instance—upsets the balance and initiates a process that can lead to a cascading failure of that ecosystem. Adding elements to an ecosystem also impacts the whole. Look at what happened when the internet gained popularity: the addition of this new technology overturned everything we thought we knew about operating a business. Cloud computing had a similarly large impact on the corporate ecosystem.

Apply Systems Thinking

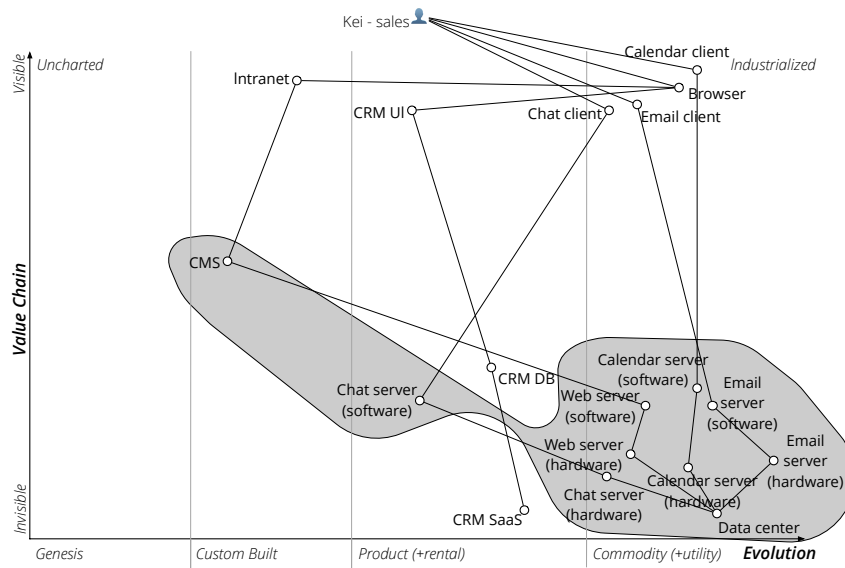
When approaching the issue of sustainability, try to think of the bigger picture. Systems thinking—a holistic overview of complex and intertwined components—allows you to get a sense of the complexity with which you're dealing and how even seemingly small actions on one piece of the puzzle may affect other pieces.⁶

For example, moving your software from on-premises to the cloud affects not only those performing that transition but also the people who need to sell and market the software as well as those who need to recruit and hire for newly changed roles in programming and IT. Even ripple effects of this change have ripple effects. It's complicated.

4. <https://sloanreview.mit.edu/projects/corporate-sustainability-at-a-crossroads/>

5. <https://www.merriam-webster.com/dictionary/ecosystem>

6. <https://www.chelseagreen.com/product/thinking-in-systems/>



Remember this diagram from [Chapter 8, Inbound FOSS in Your Corporate Strategy, on page 131](#)? Replacing all that hardware with cloud computing services may not impact Kei very much, but others in your company will certainly feel the change.

- IT no longer needs to maintain that server hardware.
 - What will become of the hardware itself? Will the company sell it? Decommission it and put it in storage?
 - What of the people who used to maintain the hardware? Will they be reassigned to other projects, or will they now be redundant and need to leave the company? If they're reassigned, have you planned budget and time for training them for the new tasks?
- Speaking of budget, switching from the capital expenditure of server hardware to the operating expenditure of cloud services subscriptions will require the people in finance to reconsider budgets.
- The recruiting team needs to learn how to locate and hire candidates with cloud computing skills.
- Procurement must track acquisition and spend differently for subscription-based services.

These are only a few of the ripple effects that a change that appears simple on paper can have once its complexity becomes evident.

Your Software Supply Chain Is an Ecosystem

Now turn your attention to your software supply chain. It's a vital element in the ecosystem of your business, but it's also an ecosystem unto itself with complex interdependent parts. If one element of that ecosystem—one link in that chain—should change drastically or even disappear, then the entire ecosystem could feel the effects.

It's not only your SSC that's affected. With a FOSS supply chain, each link in your chain is freely available to all and therefore also a part of innumerable other software supply chains and ecosystems. The SSCs of other companies are included, of course, but each free and open source project supports the ecosystem of its community and is a part of the larger ecosystem of the programming language or framework in which it's written. These components are also elements in the greater FOSS ecosystem, which similarly is a crucial part of the massive ecosystem of all software.

With this bigger systemic picture in mind, consider what sort of impact the health and longevity of free and open source software has on not only your business but every software-using concern around the world. What, therefore, is the state of FOSS sustainability?

FOSS Sustainability

Typing “open source sustainability” into a search engine returns a lot of results lately. The topic of FOSS sustainability has been all the rage these past few years, though it's usually phrased as “pay the maintainers” rather than as sustainability. As you'll learn, it's rare that any sort of sustainability—FOSS or otherwise—is as simple as throwing money at the problem.

The Situation with FOSS Sustainability

The issue of FOSS sustainability sprang to popularity in 2016 with the publication by the Ford Foundation of *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure*.⁷ This research report, written by Nadia Asparouhova (née Eghbal), shines a light on the woeful state of many critical FOSS projects and their maintainers.

These maintainers regularly work alone. In 2023 a study by Josh Bressers found that over half of all releases on the Node.js package manager (npm) show only a single maintainer for the component.⁸ That's 18 million releases

7. <https://www.fordfoundation.org/work/learning/research-reports/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/>

8. <https://anchore.com/blog/open-source-is-bigger-than-you-imagine/>

of FOSS components—and an awful lot of potentially overworked people and single points of failure (SPOF).

As any manager or business leader should know, overworked people aren't able to do their best work. Errors are more likely to creep in, and without help to review changes those errors aren't located before they're released to the users. Overwork also leads to burnout, and burnout leads to abandoned FOSS components, perhaps even some in your company's FOSS supply chain.

It's rare this overwork is the fault of the maintainer of a component. Instead, as is often the case, the problem lies in the system in which a mistreated individual is trying to operate. From the beginning of free and open source software, people and companies have used FOSS without contributing back to or otherwise supporting the communities from which they benefit. It became perfectly normal to abuse this valuable resource, demanding more work from maintainers already stretched nearly to the breaking point. After the publication of *Roads and Bridges*, people started to recognize that this development wasn't meeting the needs of the present without compromising the ability of future generations to meet their own needs. In other words, it wasn't sustainable.

Why This Matters for Your Business

Roads and Bridges—as well as several high-profile FOSS security vulnerabilities with names like Heartbleed⁹ and Log4Shell¹⁰—was a wake-up call for individuals and especially for some companies that rely on FOSS components. Previously, most organizations hadn't even realized that they had software supply chains. Now, more and more of them are recognizing not only that they have a software supply chain but also that the links in their FOSS supply chain are valuable resources that require their attention to ensure business continuity. Just as problems with the silicon chip supply chain can disrupt business for a company that builds connected devices, fractures in a free and open source SSC can disrupt operations for any software-using organization.

Short of stockpiling chips or rearchitecting the device to use alternatives, a device manufacturer may not be able to do much to avoid the effects of a chip shortage. Thankfully, in the case of the links in your free and open source software supply chain, your company can do plenty to help ensure sustainability and avoid business disruption.

9. <https://en.wikipedia.org/wiki/Heartbleed>

10. <https://en.wikipedia.org/wiki/Log4Shell>

What Your Company Can Do About It

The first step, however cliché it may sound, is to admit you have a problem. Of all of the companies with which I've worked and spoken, the overwhelming majority aren't aware of their SSC at all, let alone of the weak links in it. Once made aware, few choose to take action. More than once, I've seen one of these companies mentioned in the tech press for suffering from vulnerabilities and related issues that were easily preventable with even a small amount of intention and attention.

Since relatively few companies have paid much attention to their FOSS supply chains, the ones that do are at a competitive advantage. The odds are that their competitors aren't doing much to ensure the integrity and sustainability of the links in their chain and so are at risk. When a link disappears because of maintainer burnout, when a project's sponsoring company fails, or when security researchers discover a critical vulnerability, your company will be prepared and safe while theirs will not.

Identify and Evaluate Critical Links in Your FOSS Supply Chain

With a house, you can safely knock down some walls, but others, if destroyed, would bring the entire structure down around your ears: those are the load-bearing walls.

Similarly, and as you remember from [Reduce Scope by Focusing on Load-Bearing Links, on page 188](#), your FOSS supply chain contains what I call *load-bearing links*. Have a look at your entire FOSS supply chain. It may comprise thousands of links. While some may be unnecessary litter, by and large most of the links are necessary. But relatively few of them are *critical*. For most of the links, their loss would cause some inconvenience. For a few, their loss initiates a crisis. The chain will collapse—and your business and customers will suffer—should a critical, load-bearing link break.

What are the critical links in your FOSS supply chain? I guarantee that your list will differ from those of other companies or even from those of other lines of business within your own organization. When reviewing your SSC, look for those links that if removed could cause your entire business to suffer in some way.

Also, look for links that can't be replaced relatively easily with an alternative solution. If the replacement of a component will require a rearchitecture of the software, it's not easy. If, however, a replacement requires changes

to support a different API but provides similar functionality, it may require less work to implement.

For instance, your software development team may use an object-relational mapping (ORM) library to abstract away the details of interacting with a database. This may make swapping out database software less of a burden, yes, but it means that the ORM library may be a load-bearing link in your chain.

Don't forget: software utilities are a part of your supply chain and can also be critical links. These are often overlooked because they aren't a direct component of the software, but without these utilities the software may be impossible to create. A good example of this is the Free Software Git version control system (VCS). It's the most common VCS used today (I even use it for this book!) and is the source of truth for the projects for which it's used. It allows software developers to collaborate, and it's the foundation on which many other development processes are based. It, and utilities like it, may be critical free and open source links in your software supply chain.

Reviewing and identifying critical links in your SSC is a cross-functional effort. A group of managers sitting around a table looking at component lists and architecture diagrams are unlikely to have enough information on their own to determine which links are important enough to require special attention. Involve people from security, IT, technical program management, programming, and any other relevant function. Get their input during the process, and listen to it. After all, if something goes wrong with a link, then these are the people who will be most directly affected in fixing it.

Assess Health and Needs of Those Critical Links

Now that you have a list of critical links, you can dive in and start providing sustaining support to these components, right? Wrong. Your enthusiasm is great, but cool your jets, turbo. You don't even know what kind of support these links need or if they even need support at all. *Don't* make assumptions about the health and needs of FOSS projects and communities. *Do* take some time to learn about their health and needs.

Project Health Is a Sustainability Indicator

Project health is a great indicator for how sustainable a FOSS component may be. You can look at a lot of factors to gauge the health of a FOSS project and community, but these are some of the most common:

- *Activity*: Is anyone working on the project at all? When's the last time they updated code or bug reports?
- *Responsiveness*: When someone asks a question, does it languish or does someone respond within a day or so?
- *Number of contributors*: Is this project maintained by one person? Or are there many different contributors who can help carry the burden of maintenance?
- *Disposition*: When the project receives questions or contributions, do they respond rudely or are they welcoming and helpful?
- *Code of conduct*: Similarly, does the project have and enforce a code of conduct, ensuring a welcoming environment for all community members?

Many more things can factor into the health of a free and open source software community, but you usually don't need to go into much more detail than the preceding items to learn enough for your initial sustainability effort. If you'd like to learn more, the Community Health Analytics in Open Source Software (CHAOSS) project maintains resources defining nearly 80 different metrics by which you can measure FOSS community health.¹¹

Don't Assume You Know Better Than the Project

Ask anyone who uses a wheelchair and they'll tell you that one of the things that irritates them most is when someone grabs their chair and starts pushing them without asking. However well intentioned, it's arrogant and insulting to assume you know someone's needs better than they themselves do, and FOSS communities are no exception.

In your research about a project's health you may *think* you know what a project needs, but you lack the perspective of a community member and therefore may be incorrect in your assumptions. If you want to know what a project needs, ask them. It doesn't take much to do and it won't hurt, I promise. I'll even give you a script you can copy and paste into an email:

Hello! I'm from ORGANIZATION. We use and rely on your project and are really grateful for all the work you've put into it so far. We'd like to help support the project in some way. What are your pain points right now? What would make the biggest positive difference for the community? Maybe there's some way we can lend a hand with that.

You may find that some projects are healthy and don't have many immediate needs. That's great! These are critical links that you can keep your eye on in

11. <https://chaoss.community/kbtopic/all-metrics/>

the long term but don't need any immediate action from you or your company, freeing up resources for the projects that need it more. After all, as much as you may wish to, you can't support every single link in your free and open source software supply chain, not even all the critical ones. By learning more about the health and needs of a project, you can make informed judgments and make the biggest difference with your undoubtedly limited resources.

Outbound Contributions

As you research the needs of your critical projects, you may learn that what a project needs most is more people to help with the work of building and maintaining the component on which your company relies. Maybe it's a single-maintainer project and that one person is on (or over) the edge of burnout. Maybe it's a larger project with multiple maintainers but still not enough people to keep up with the workload. Regardless of the situation, your company can help.

Work with the project maintainers to learn where they need more help, then dedicate time from your own staff members to fill that need. Lots of roles are necessary for successful software development, so don't assume that programmers are the only staff members who can lend a hand. Some of the many tasks with which the project may need assistance include these:

- Triaging problem reports
- Managing bug queues
- Writing documentation
- Performing security audits
- Writing and fixing tests
- Fixing bugs and adding features
- Writing and editing blog posts
- ...and more!

Providing this sort of help is known as making *outbound contributions* to the FOSS project: contributions originating inside your company but directed outward to the FOSS project. You'll learn all about outbound contributions in [Chapter 17, Master Contribution Fundamentals, on page 263](#).

Hire Maintainers

Among your list of load-bearing links in your FOSS supply chain may be components that are particularly strategic. Maybe the component provides functionality that's vital to your business, or perhaps the company has plans to build a product offering around that component. It could be that your company is paying a vendor a great deal of money for (mostly unused)

technical support for the project. Whatever the reason, it's an especially important link in your software supply chain.

For FOSS projects like this, it can make good business sense to hire the maintainers to be staff at your company and work full-time on the project. This allows them to continue improving the FOSS project for everyone while also, when needed, providing technical support internally and training others to do the same. Your company could then cancel the expensive support contract, transferring that budget to pay the salary of the maintainer. By taking on a full-time role working on their FOSS project, the maintainer gains the peace of mind that comes with a stable salary (as well as healthcare insurance if they're in the United States) and all of the mental health benefits that come with it. The overall picture becomes one of stability and sustainability.

Something to consider, should your company choose to hire maintainers, is you may not want to assign them into the normal software development reporting chain. I've seen companies make this mistake all too often, resulting in the maintainer being diverted onto full-time product work rather than the strategic component for which they were hired. This frequently leads to resentment on the part of the maintainer, who may then leave the company, moving to one that will allow them to return to working on the FOSS project.

You'll find many different ways to slice the apple of team composition, and some ways may work better in your organization than others. In this case, it can make sense for people like this to report into the OSPO so that all major open source initiatives are managed through a single team. See [Chapter 6, Introducing the Open Source Program Office, on page 101](#), for more information.

Foundation Membership/Participation/Support

Like any other human endeavor, as it grows a FOSS project gains administrative overhead. On top of the already considerable work of maintaining the software and its community, this “administrivia” becomes a burden that many FOSS projects would rather be managed by others.

Thankfully, some nonprofit organizations (NPOs) and foundations are formed specifically to perform these tasks. These NPOs act as the administrative home base for FOSS projects, managing their finances, providing legal assistance, supporting infrastructure, and performing other duties as their member projects require. Often, the NPOs specialize in certain domains. For example, Software Freedom Conservancy prefers member projects that further software freedom, the Apereo Foundation supports FOSS projects related to

higher education, and the Open Bioinformatics Foundation specializes in FOSS and open science initiatives in biological research.

Depending on the needs of critical projects in your FOSS supply chain, it may make sense for your company to join or participate in the foundations that support those projects. Free and open source software is not pay-to-play; your company will *never* need to pay or join a foundation to use or contribute back to any FOSS project or to join its community. It may, however, choose to join one or more of these foundations as sponsoring members to provide additional support and sustainability for projects that are critical to its business operations.

At the very least, it should consider sponsoring fundamental foundations for the technology that enables those operations. Companies that rely on the Python programming language should support the Python Software Foundation, those that use the FreeBSD operating system should support the FreeBSD Foundation, and all companies that rely on free and open source software should support the Open Source Initiative. Fundamental technology foundations like these ensure that the overall FOSS ecosystem continues to grow and thrive, but they also require our support to continue operating.

Monetarily Sponsor Projects

It's less common than some people would have you believe, but sometimes the answer to sustainability for critical links in your FOSS supply chain is to throw money at the problem. If nothing else, it's absolutely the easiest way for a company to feel it's done its due diligence where FOSS sustainability is concerned.

Many FOSS projects provide ways for companies and individuals to donate money to them. Some have signed on with a mission-driven nonprofit fiscal sponsor, such as Software Freedom Conservancy (SFC), to accept and manage donations on their behalf. Because these donations officially go to the nonprofit (though earmarked for the project), in many jurisdictions these donations are tax-deductible. Other projects have joined trade associations like Open Source Collective. These associations also accept and manage donations for the projects but are not the sort of nonprofit organization to which donations are tax deductible. Projects may also accept donations via for-profit organizations such as PayPal, Patreon, or via initiatives like GitHub Sponsors.

Whether they've established a payment mechanism or not, all too often projects receive money but aren't really prepared to do so. I've heard from several

project maintainers who suddenly found themselves with thousands of U.S. dollars in unexpected donations. Of course, they were elated; who wouldn't be? Free money! Once the celebrations subside though, the maintainers discover they have additional burdens that are just as unexpected:

- Where should they keep the money? If it's a multi-maintainer project, an individual's bank account may not be the right answer, so what is?
- These donations may count as income and therefore may bring tax implications with them. Are the maintainers aware of those implications in their jurisdiction?
- How will they even use the money? They could split it up across maintainers, but how do you decide whom to pay and how much?

These are only a few of the questions and burdens that project maintainers face when unexpected money lands in their laps.

So before taking the easy way out by opening a purchase order or pulling out the company credit card, take a moment to confirm that your donation won't cause more harm than good. Projects that have signed with a fiscal sponsor like Software Freedom Conservancy will have all their financial ducks in a row, thanks to the services provided by Conservancy to its member projects, and are entirely safe to donate to. For others, at the very least make sure that the project has a valid way to accept payments. If you're able, speak with the project maintainers and see whether there's a specific line item for which your company can pay. This relatively small step can do a lot to answer a whole lot of questions where project sponsorship is concerned.

Make It a Priority

Whatever method (or, more likely, methods) you choose to sustain the critical links in your business's free and open source software supply chain, you also need to consider the sustainability of the effort itself. In any organization, unprioritized projects have no chance of success; something will always be perceived as more important.

As you start your journey toward SSC sustainability, work to gain support from upper leadership. Using what you know of your company's business needs and what you learn in this book, present a solid business case for the effort. Supporting the FOSS projects on which your company relies is the right thing to do, but don't depend on altruism to convince people. Show how this initiative will benefit the business.

Develop a long-term plan for FOSS supply chain sustainability. The entire point of making things sustainable is to ensure longevity, and you can't do that with a plan that's a one-and-done sort of thing. A good way to do this is to make free and open source software sustainability a part of the overall corporate sustainability plan. The links in your supply chain are a valuable resource, like any other, and require the same strategic attention if the company is to remain a going concern into the future.

The strategic importance of the critical FOSS links in your software supply chain reinforces the necessity for knowledgeable people dedicated to delivering on the promise of the planned effort. Just as you wouldn't trust your company's financial operations to those without deep knowledge of budgets and balance sheets, you shouldn't trust FOSS sustainability to well-intentioned but unskilled staff. Therefore the company's Open Source Program Office (OSPO) is key for the success of any FOSS sustainability effort. These staff have the knowledge and experience required for navigating the tricky course through the various FOSS communities and are committed to building and maintaining the necessary relationships for sustaining these load-bearing links in your company's FOSS supply chain.

Part III

Outbound (Contributing to and Releasing) FOSS

The third and final part of the book is all about contributing to and releasing free and open source (FOSS) software, aka outbound FOSS, starting with laying a solid foundation for outbound FOSS, related policies, and strategic reasons for outbound engagement by your organization.

Next, you'll discover ways your organization can contribute to FOSS projects and communities that are effective for both sides of that relationship.

Releasing FOSS projects can be a powerful tool for any company, but only if they do it correctly and for the right reasons. You'll learn all about this next.

Finally, you'll learn you can make a living from giving away software by using business models of successful open source-based companies.

Level Up Your Business with Outbound FOSS

In [Part II, Inbound \(Using\) FOSS, on page 121](#), you learned how inbound FOSS can help (and sometimes harm) your business. In this final part of the book, you'll take a little walk outside of your company into the world of *outbound* FOSS. Like inbound, outbound FOSS holds the potential for brilliant strategic business moves, but it also hides some risks that can damage your brand—or worse. The following chapters teach you how to engage in the external FOSS world, sidestepping risks while being a model corporate FOSS citizen.

What Is Outbound FOSS?

The obvious first step on this journey is learning what's meant by outbound FOSS (often referred to as simply “outbound”). It's any sort of FOSS engagement outside of the confines of your company. This includes contributing to FOSS projects, supporting those projects and communities, releasing FOSS projects and resources developed by your company, and managing and participating in the communities of your company-originating FOSS projects.

Outbound Impacts the Entire Business Organization

As you learn about outbound FOSS in the following chapters, keep in mind whom your outbound initiatives will impact and, therefore, whom you must include in your planning and implementation.

Usually people think outbound only affects the software developers in the company because they think of free and open source only as software and

not the strategic asset that it is. The more enlightened companies I've seen may expect those effects slosh over into marketing a little. In truth though, when handled correctly and strategically, outbound FOSS initiatives count several different teams among their stakeholders.

Since outbound involves making its intellectual property available outside of the company, Legal is an obvious stakeholder in these efforts. You'll need them to be involved not only to ensure that the IP is handled correctly but also to keep the company's risk and potential liability at a reasonable level.

Compliance and Security play similar roles to Legal in outbound. They make sure that everything is secure and that the company doesn't take on any undue risk in its outbound efforts or that it doesn't release projects containing vulnerabilities.

The documentation team is critical to the success of any outbound FOSS. They can capture and communicate the internal processes for outbound engagement, and they can also make certain that any FOSS projects or contributions released outside of the company are useful to the target audience. Nothing helps a company's brand quite like trusting and caring enough about its users to provide effective documentation.

The recruiting team can use the company's outbound engagements to locate qualified candidates more quickly. It also will find that a lot of those candidates will find that the freedom to contribute to FOSS for the company is a tempting perk of working for it.

Over in Product, they'll have a large and engaged pool from which to do market research and user interviews. More often lately, companies have product offerings that are either based or heavily rely upon specific FOSS projects. When the evolution of a project directly impacts the direction and customer experience of a product offering, it becomes vital that the product team stays informed and involved with that FOSS project evolution.

The marketing team is used to thinking externally to the company, which will be useful in many different ways. They not only can promote your outbound initiatives, they also will have leads on potential markets for them and knowledge about how to speak to and reach those markets.

Naturally the programming and software development teams will need to be involved with anything involving code, but that's not where their engagement ends. If your company chooses to release FOSS projects, someone will need

to manage the resulting projects and communities.¹ Often, the best people for these jobs are already a part of your software development team.

This list is only scratching the surface of how widespread your outbound FOSS engagement can be within your organization. For instance, with the help of your Documentation and Internal Comms folks you can develop and share the resources that can allow anyone in the company to contribute and engage with your outbound FOSS initiatives. Finance, artists, designers, IT, translators, accessibility experts, everyone can make a positive difference. Staff members across the entire organization can engage with outbound FOSS in an infinite variety of ways, if only you'll include and support them.

While reading the upcoming chapters, consider whom else you might want to involve or consult as you develop your outbound FOSS strategies and plans. The earlier you include them in the conversation, the better chance you'll have of meeting your goals.

Outbound Isn't Limited to Code

Just as business leaders usually think of programmers as the only or primary stakeholders in outbound FOSS, they also usually think of it as only code-based. However, as you probably already figured out from the previous section, it can involve a lot more.

Outbound can encompass so much more than simply code and software. Creative works are ripe for releasing as open projects under Creative Commons licenses. Icons, stock photography, fonts, music, videos, documents, even entire books can be valuable tools in your outbound FOSS strategy toolkit. For example, IBM released their official font, Plex.² The New York Public Library shared the vocabularies and mappings it uses for the ontologies that help organize the entire collection.³

Beyond the things the company creates—be they code or creative works—outbound initiatives can also include the skills that do that creation. Providing accessibility or security reviews, translating documentation and user interfaces, administering IT infrastructure, managing projects and communities, mentoring contributors, and guiding project governance are some of the many different skills that exist in your organization and could be valuable contributions to external FOSS projects.

-
1. <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>
 2. <https://github.com/IBM/plex>
 3. <https://github.com/NYPL/nypl-core>

Remember, this is free and open source *software*. No software project, FOSS or not, succeeds without people playing many different roles. Aside from the obvious programmer role, also needed are the system administrator, quality assurance tester, customer support representative, technical writer, accessibility auditor, designer, and many others. The options for outbound engagement are only limited by your strategic imagination. Thinking outside of the code-and-software-only box expands those options in ways you'd never have imagined before.

Why Do Outbound FOSS?

Why put in this work? Many companies have found that actively supporting employee efforts to participate in outbound FOSS is a good way to increase staff retention,⁴ decreasing workload on its already busy recruiting team and boosting productivity through continuity. It's also a great way for those employees to get professional development and build valuable networks while also supporting the FOSS projects that are important to the business. A 2021 survey from IBM and O'Reilly Media showed that the respondents feel skills in FOSS lead to higher wages and more job opportunities.⁵ Don't doubt that if your company doesn't support professional development through FOSS, plenty of other companies out there will. Don't risk losing your valuable staff members by ignoring this avenue for retention.

Now that you have an understanding of what outbound FOSS is, it's time to set up some guardrails so your company can travel the outbound roads safely. It's time to talk about outbound policies and procedures.

4. <https://builtin.com/recruiting/use-open-source-software>

5. <https://developer.ibm.com/blogs/oreilly-open-source-skill-survey-blog/>

Outbound Policies and Procedures

Before you unleash an outbound FOSS strategy on unsuspecting staff, make sure you have your outbound policies and procedures ready to go. Set your people up for success by providing the information they'll need to answer their questions so they can do the right things at the right time for the right reasons. You'll do that by building upon what you learned in [Chapter 9, Establish Effective Inbound Policies and Procedures, on page 145](#), and [Chapter 10, Get Started with Your Inbound Policies, on page 163](#), to add outbound FOSS into the mix. By the end of this chapter you'll have a good idea of what your company's complete FOSS P&P should look like.

Outbound FOSS Policies

It's likely that you'll consolidate your inbound and outbound FOSS policies into a single policy document, and that's OK. The only wrong way to do it is not to have inbound or outbound FOSS policies at all. In this book, however, the two are considered separately for ease of locating the information you need when you need it.

IP Policy

As is often the case when discussing FOSS policies, intellectual property (IP) is usually at the front of everyone's minds. This is understandable and appropriate, considering [IP Is the Foundation of FOSS, on page 17](#). As IP is the basis on which FOSS is built, it makes sense that your company's outbound IP policy will become the foundation for its other outbound FOSS policies and procedures (P&P).

Your organization's outbound FOSS IP policies explicitly state how employees can use and otherwise manage the company's intellectual property outside of the company itself and whom to contact should they have any questions.

These IP policies are especially common in larger companies that receive a significant portion of their revenue from licensing company patents, but all companies that engage in outbound initiatives should create and maintain them. The persons typically responsible for creating, maintaining, and enforcing these policies are the fine folks in the company's legal team, so make sure they're involved from the very start.

Does your company already have a set of policies for intellectual property? If it's been a while since you've looked at them (or if you never have), grab a tasty warm beverage and revisit them now. In recent years it's become more common for these policies not only to exist but also to include a section specifically about the use, contribution, and release of company IP for free and open source software. Being familiar with the current state of your company's IP policies will be helpful as you work through the remainder of this book. Doing that will prepare you for the thinking you'll need to do about outbound FOSS in your company, and it also may reveal that the policies need changing to support any planned strategic inbound or outbound FOSS engagement.

Don't Rely Solely on Employment Agreements

Sometimes you'll find that the company's IP policy primarily exists in its employment contracts or agreements rather than in a separate document. While it's certainly a good idea to make sure all employees know about and agree to the IP policy as they join the organization, I recommend companies not rely solely on policies enshrined in employment agreements. These are documents that people read once (maybe) then sign and move on.

The agreement applies for the duration of an employee's tenure at the company, but rarely are they revisited or even remembered. Let's be honest here: how many of the employees in your company read their employment agreement, and how many of those are able to wade through the dense legalese to understand what they're agreeing to? Or remember it for the duration of their years of service to the company?

In many ways, employment agreements are a lot like those End User License Agreements (EULA) or Terms of Service (ToS) popups to which we're all subjected: we're trying to get something done, we can't do it until we agree to check a little box and click Submit, so we do that so we can get on with our lives. We've just agreed to legal terms without reading them, since we've learned that we have little choice to do otherwise. Negotiating the terms is unlikely to work, after all, and would delay us from the thing we need or want.

Instead of entrusting something as critical as company IP to a document that few read or understand, I recommend you have a separate, stand-alone IP policy that's written in plain language, making it easy to understand and to find whenever people need it. Relying on IP policy via employment agreement is inadequate and can put the company's intellectual property at risk of (usually) unintentional exposure by employees.

In contrast, a separate IP policy acts as valuable reference material, and being more modular, it's also a lot easier to change if needed. For instance, does your organization wish to change its IP policy to give employees more freedom to contribute to FOSS? While it's certainly not impossible to modify or amend a policy that exists only in an employment agreement, it can be complicated.

Avoid that complication entirely by having the policy be a separate work that's referenced in the employment agreement rather than embedded within it. The company can then update the IP policy as needed, consulting stakeholders in the process and notifying everyone after finalizing the change. The stand-alone policy also makes it easier to create training modules for the policy and to promote the policy via internal communications.

What Should Be in Your Outbound FOSS IP Policy

While, as you'll read about shortly, some companies publicly post their policies, resist the urge to cut-paste one of these and then use it as-is. For one thing, the license of the content may not allow this (or at least not without attribution), but also that policy is unlikely to meet the business needs of your own company. Certainly use example policies to give you an idea of where to start, but your company's policies must be unique to the problems it faces in its business and operations.

As you're drafting the policy, try to focus on concepts more than implementation. This is a policy, not a procedure. Discuss what people can do, under what conditions, and most importantly *why* the policy is as it is. Remember from [Communicate, Communicate, Communicate, on page 155](#): you hired smart people. Trust them with the reasoning behind the policies so they can make informed decisions.

The policy should discuss three areas, which you'll learn about in more detail in the sections that follow:

1. Making outbound FOSS contributions
2. Releasing internally developed projects as publicly available FOSS
3. Where to get help with questions, exceptions, or edge cases

Policy for Contributing to FOSS

The most obvious question to answer with a FOSS contribution policy is are employees allowed to contribute *at all*? For companies in highly regulated industries, the answer may be, “NO, NO, NO.” Whatever the answer, it’s important to make it clear and easy to locate (though perhaps phrase it more nicely and not in all-caps).

If employees are allowed to contribute, under what conditions and what form may those contributions take? For instance, is it OK to file a ticket if the employee locates a bug in a FOSS project? If it’s OK to file a bug, is there anything they shouldn’t include in that report, such as company source code? If the bug report does require source code to show where the problem is happening, does the employee need to get permission from their manager to file that bug ticket?

Speaking of conditions, don’t forget the copyright ownership matrix (reproduced here in the following figure) in [Work for Hire and Its Effect on Copyright, on page 29](#). Make sure the FOSS contribution policy is clear under what conditions an employee owns copyright to a work and what conditions mean the copyright belongs to the company. Use the matrix in your policy if you feel it will make copyright ownership clear for everyone. Some people do better with pictures than with text, after all, so accommodate that in your policies to ensure everyone can understand them well.

		Equipment	
		Employee's	Company's
Time	Employee's		
	Company's		

Another matter to be clear about is what people should do if a FOSS project requires them to sign a CLA ([Contributor License Agreements, on page 34](#)) or use the DCO ([Developer Certificate of Origin, on page 35](#)) to make a contribution. Each of these agreements requires the contributor to attest that

they either own the copyright over their contribution or have permission from the copyright owner (your company) to share the copyrighted material. Where do they get permission? Are they allowed to sign a CLA on behalf of the company? I bet you a cookie that your legal team will have strong opinions on that last question in particular. Save everyone a lot of headaches (especially that legal team) and answer these and related questions up front in your outbound FOSS policy.

Finally, while the steps for how to contribute should not be a part of the policy, you should definitely provide guidance on where to find that procedure. At all points and in all policies and procedures, make it as easy as possible for your staff members to do the right thing. Remember, if you don't provide guidance on where to find the procedure, your enterprising employees may make those steps up for themselves, and you may not like what they come up with.

Open the Doors to Open Source Contributions

In 2020, the FOSS policy-making world received gifts in the form of the GitLab Open Source policy and the GitHub Balanced Employee IP Agreement (BEIPA). Both of these companies are known as destinations for others to visit to contribute to FOSS projects, so they wanted to enshrine their commitment to open source in their IP policies.

These SaaS software development system companies each changed their policy about contributing to FOSS projects to streamline the process for their employees, giving them permission to contribute in most of the situations that the staff members would come across in their daily work lives. Not only did they change their policies, but—true to the spirit of FOSS—they made sure that other companies can benefit from their work: these policies are available for all to view and repurpose.

- The GitLab policy—as well as its entire employee handbook—is available under a highly permissive MIT license.^a
- GitHub made their BEIPA public domain, releasing it under a Creative Commons Zero license.^b

As you work through the outbound FOSS drafting process, keep these examples in mind. Perhaps the best policy for your company's needs is to open up your open source contributions.

a. <https://about.gitlab.com/handbook/engineering/open-source/>

b. <https://github.com/github/balanced-employee-ip-agreement>

Policy for Releasing FOSS

Similar to contributing, the primary question to answer for releasing projects as FOSS is whether people in your company are allowed to do it at all. After reviewing the company's goals and available resources, you may have determined that releasing and maintaining FOSS projects and communities will be too much of a drain on people's valuable time. That's understandable, but as you'll learn in [Chapter 19, The Strategy of Releasing FOSS, on page 297](#), you can gain a lot from releasing projects, strategically speaking.

Assuming it's OK for folks to consider releasing projects, what sort of things are fair game? Usually, specifying rough categories and criteria can give people some idea of whether they should even entertain the thought of releasing something as a free and open project, software or not. For instance, any sort of materials that contain business logic may not qualify as releasable, but supporting utilities and content may.

Like most matters around setting a FOSS policy, these categories are unique to your company and its business needs. You'll need to put some thought into it, but don't overthink it. Start with a low-fidelity policy, with criteria roughly sketched out, and require people to verify permission to release for all projects. Do this for a little while and you'll get a sense of the sorts of projects people are looking to release. With this real-world information, you can then refine the policy as needed.

As should be the case with all policies, this one should include pointers to where people can find the procedures for publicly releasing projects under free and open licenses. Always make it easy for people to do the right thing.

The Edge Cases

In the previous section you learned to start with a roughly sketched but still usable policy rather than overthinking it from the start. This is good advice for any policy, not only those related to outbound FOSS. Avoid making the initial policy too specific at the start. Put enough thought into it to make it generally applicable and add just enough friction so you can keep an eye on how people are using the policy. In the preceding example, that friction takes the form of requiring permission to release projects.

No matter how many times you refine a policy, or even if you overthink it, edge cases will always turn up that the policy doesn't cover. This is OK! It shows how clever and innovative people can be, and that's what you like to see in a

business. Also, don't forget that technology doesn't care about your policy; it'll keep right on evolving.

Only a few years ago, distributing software via containers was a new idea, bringing with it a lot of shiny new edge cases for FOSS policies (visit [The Container Complication, on page 184](#), to refresh your memory on that topic). Now it's commonplace for a FOSS component to include a container as one of the installation options. Edge cases evolving to the commonplace is precisely why you should always be ready to [Evolve the Policies, on page 160](#).

Regardless of the policy, always give people an easy way to ask questions and receive answers. This could be as simple as an email address to which they can write. If you go this route, I advise you to make it a *role address* rather than a personal one. Instructions to email `a.obeng@bees4less.com` only work for as long as Adric Obeng works at the company. When he leaves, someone needs to remember to update the policy, which we all know is unlikely to happen. Instead, have IT create a role address along the lines of `oss-help@bees4less.com` and have multiple people receive the messages sent to it.

Even better, have the policy send their questions in via a ticketing system. Opening a ticket not only gives people a way to ask questions, it also creates accountability through ticket assignments and an archive of questions asked in the past. This is a valuable resource when you [Evolve the Policies, on page 160](#).

Whatever the method of handling questions about edge cases, for FOSS policies your Open Source Program Office (OSPO) will be the best team to provide answers. These people are the experts on FOSS communities and on your company's FOSS-related policies and should be able to answer questions quickly and efficiently. For more involved or "edgier" edge cases, the OSPO is well positioned to coordinate discussions among the legal, security, IT, or other stakeholder teams. Revisit [Chapter 6, Introducing the Open Source Program Office, on page 101](#), to learn more about the OSPO.

Example Outbound FOSS Policies

At this point you're probably eager to see example outbound FOSS policies. You've already been introduced to two of them in [Open the Doors to Open Source Contributions, on page 237](#). Thanks to the open nature of free and

open source software communities and practitioners, many more examples are available online. For instance, several companies make their FOSS policies publicly available:

- Microsoft¹
- Citi²
- Zalando³

You could search the internet for other examples, but you'll be glad to learn that someone has already gathered them for you. The TODO Group, an OSPO community of practice hosted by the Linux Foundation,⁴ maintains a collection of publicly available corporate FOSS policies.⁵

As you develop and introduce your company's FOSS policies, consider drafting them in such a way that they also can be made public, then add them to the TODO Group list to help others with their own FOSS policy-drafting challenges.

Outbound FOSS Procedures

As you're working through this part of the book and developing your outbound initiatives, simultaneously develop the procedures necessary to implement those initiatives. Ideally, you'd be able to coordinate with program or project managers and technical writers to do this, but not everyone will be lucky enough to have that kind of professional assistance available. You will, at least, be collaborating and consulting with various stakeholders during the process of creating the strategy. As you do so, ask them what's required to make that strategy a reality. What tasks must someone perform? Work with the stakeholders to ensure that procedures for all necessary tasks exist and are accessible prior to launching your shiny new outbound FOSS initiatives.

Procedures to Consider

What procedures will your company need for supporting its outbound FOSS policies? Just as the policies are unique to your company, so too are its procedures. It makes no sense creating procedures that aren't tailored to the policies, after all. The policies of your company provide the *what* and the *why* for a matter, while the procedures provide the *who*, *where*, and *how*.

-
1. <https://opensource.microsoft.com/program/>
 2. <https://github.com/Citi/citi-ospo/blob/main/FOSS%20Contribution%20Guidelines.md>
 3. <https://opensource.zalando.com/docs>
 4. <https://todogroup.org>
 5. <https://github.com/todogroup/policies>

The details will vary by company, but nearly all of them will need procedures that present guidance for the following topics:

- How to contribute to a FOSS project
- Approved list of projects to which to contribute (no asking necessary)
- How to handle a project having a CLA or DCO
- Whom and how to ask for permission to release a project
- Steps necessary before a project can be made public
- How to plan for and then actually manage a FOSS project you've released
- And always, where to go if you need help or to escalate a problem

As you review the example policies in the preceding list, you'll see how these disparate companies approach these topics, and you'll get inspiration for how to address them for your company's business needs.

Postponing P&P Is a Mistake

Developing outbound FOSS P&P may seem like a step you can skip or say “oh, we'll get around to it later,” but I guarantee you'll regret it if you do. Far too many companies—especially software startups—ignore their P&P entirely, potentially leaving the company IP and security at risk. Creating policies and procedures in tandem with the strategy and tactics they'll support not only ensures they'll exist at all but also helps make sure that they'll be fit to purpose and useful.

Remember, if your organization doesn't provide P&P for how to implement its strategy, the employees you assign that implementation will make up their own rules for doing so, and you may not like what they come up with. For more information about creating P&P that work, revisit [Chapter 9, Establish Effective Inbound Policies and Procedures, on page 145](#).

Now that you have some idea of how to get started with your outbound FOSS P&P, it's time to learn about the many benefits (and a few risks) of your company engaging in outbound free and open source initiatives.

Strategic Benefits and Risks of Outbound FOSS

You've already learned a lot about corporate FOSS strategy in [Chapter 8, Inbound FOSS in Your Corporate Strategy, on page 131](#), including how FOSS can lower costs and speed digital transformation. Now it's time to add the next layer to your strategy: outbound free and open source engagement.

These are the two main types of outbound FOSS:

1. Contributions
2. Releasing materials under free/open licenses

Outbound can be as powerful a tool in your strategy toolbox as inbox, but like hammers and screwdrivers, contributions and releasing FOSS are two different tools that need to be used in different ways to get the results you want.

Be Clear About Your Goals

A lot of questions can come up when discussing outbound FOSS in the context of your business, and most of their answers can be strategic if you put your mind to it. Determining how and where your company will engage can create new business strategies or support the ones already in place. Importantly, answering these questions gives you the opportunity to document why your company is engaging in outbound FOSS at all.

By making the expected or desired outcomes—that is, the goals of outbound engagement—explicit, your company is well on the path to success with its FOSS strategies. I've seen surprisingly few companies that have expressed or documented their outbound FOSS goals; most rely instead

on vague statements that are unsupported by valid business reasons. Unfortunately, this behavior tends to parallel how most companies handle *all* of their business strategies, but that conversation is thankfully out of scope for this book.

The lack of well-expressed goals makes it difficult if not impossible for a company to be successful with its outbound FOSS initiatives. It has no way to determine whether it's on target for the goals, because there are no goals to target. The company also has no way to gauge or prove the business value of these initiatives. The end result is that, often, the outbound initiatives end up on the chopping block, even if they're doing a good job. Being clear about initiative goals up front and tying them directly to overall company strategy can ensure that your outbound FOSS engagement doesn't waste money, time, or goodwill.

Initiative Value: If You Know It, Show It

Teams that aren't able to prove a history of providing value to the company are likely to be seen as unnecessary or, at best, nice-to-haves and will probably be cut. If you want to preserve the jobs of you and your team members, learning how to make and support a business case for your initiatives is an essential—and perhaps even existential—skill.

Never forget that the outbound FOSS initiatives in your company are *business* initiatives. They use company resources with the aim of strengthening the company or its business in some way, supporting the goals and strategies of the organization. How do your initiatives do this? Do you know? Can you prove it? If not, then get cracking; you have some work to do.

The first step is to review (or discover) the chain of goals from your team to the top of the corporate ladder. Your team's goals should support the goals of your leadership, whose goals should support the goals of *their* leadership, right on up the chain to the head of the company. If your organization uses Objectives and Key Results (OKRs), then you may already have some insight into this chain of goals.

Next, determine which of those goals your team's initiatives support. If the answer is “none,” then you have a big problem with alignment between your team and the company at large. Usually though, each initiative will support at least one of the goals in some way.

Now your task is to define *how* your initiative supports the higher-level goals of the organization. It's helpful at this point to start “speaking” the language

of the company: presentation slides (PowerPoint or what have you). Prepare a report or deck laying out the following (at a minimum):

- The initiatives your team is working on
- The costs involved for each initiative in both the short- and (if it's a long-running initiative) long-term
- The anticipated and actual value provided by each initiative, expressed numerically and associated directly with one or more higher-level goals

You may need to collaborate with Finance, HR, Product, or other teams to gather this information. You also may need to use outside resources such as the CHAOSS Project¹ to learn how to gather and track other relevant metrics.

Since you'll be speaking the company's language, you probably should include relevant charts and diagrams to make it easy for leadership to see that the numbers they care about are moving in the desired direction.

You also should start the entire report with a paragraph of the report findings right up-front. This executive summary, elevator pitch, TL;DR, or whatever you feel like calling it is respectful of your audience member's time (and realistic since it's unlikely many of them will read the entire report) and will make it easier for them to follow along with your team's progress.

Once you've presented the report to your leadership and received their sign-off—confirming your team is in alignment with the current company goals—don't let the matter rest. Regularly revisit the report, updating it with the latest information and speaking about it with your leadership. You want to keep the value your team provides top of mind for the decision-makers of the company, never giving them cause to wonder what it is you do or why your team exists.

The preceding steps are phrased in terms of existing projects, but they apply just as much to any new initiative that your team may wish to launch. It's critical to provide a business case for these initiatives, since “it's the right thing to do” holds little business value (unfortunately), even if it's true. Make the case, provide and prove value to your organization, and guarantee job security for yourself and your team.

1. <https://chaoss.community>

How Outbound Contributions Can Benefit Your Strategy

Contributing to external free and open source software projects is one way to make a strategic impact, but as you'll learn in [Chapter 17, Master Contribution Fundamentals, on page 263](#), you can contribute in a lot of ways, and most of them have nothing to do with writing code. In fact, my book *Forge Your Future with Open Source* is entirely about how to contribute to FOSS, with very little of the content specific to programming.²

In the sections that follow, you'll find a few of those types of outbound FOSS contributions, which we'll cover in more detail in [Chapter 17, Master Contribution Fundamentals, on page 263](#). While they represent different types of engagement, they all share one thing: they involve people outside of the company, so communication will be key to success. This may mean involving community managers, technical writers, or others who have the skills needed to communicate effectively. Keep this requirement in mind as you read on and learn a few of the ways that outbound contributions can make a difference in your business strategy.

Sustaining the Supply Chain

Remember the company from [Inbound FOSS and the Digital Transformation of Bees4Less, on page 132](#)? Bees4Less provides a variety of bees and bee-related products for all your pollinator needs. To do this, the company needs access to flowers. A *lot* of flowers. In [Reduce Scope by Focusing on Load-Bearing Links, on page 188](#), you learned about load-bearing supply chain links. For Bees4Less, flowers are a critical, load-bearing part of their supply chain; without them, the entire business buzzes off to greener pastures.

Bees4Less doesn't sit back and hope their flower suppliers will continue providing enough blossoms for their bee inventory. They ensure a healthy supply by tending gardens and providing support to the gardeners who grow their flower stock. They also support local garden clubs, enriching the community and the environment while supplementing their stock of flowers. They sustain their flower supply chain, similar to the software supply chain sustainability you learned about in [Chapter 13, Sustain. Maintain. Retain., on page 213](#).

Your company's software supply chain undoubtedly contains FOSS components. Are any of them critical, load-bearing links in the chain? If any of those

2. <https://fossforge.com>

components were to break, be compromised, or otherwise disappear, what would happen to your business? How would it affect your customers?

It could get pretty ugly, that's for sure, but by building a program of outbound FOSS engagement that includes contributing to free and open source software projects and communities, your company can ensure the integrity of its software supply chain. By releasing internal projects as FOSS, it also can build up new links in the chain, giving them the strength needed to support upcoming product offerings. Sustaining your FOSS software supply chain through these outbound engagements guarantees your strategies more success than if you left the supply chain health to chance.

Reputation/Brand

Does your company have a corporate giving program, match donations to not-for-profits, or offer leave for volunteering in the community? They do these things not only because it's right and good to give back to the world but also, beyond any altruistic reasons, because it's proven to be good for business.³ Corporate philanthropy contributes to a positive company image, building trust with the market and strengthening your brand.

Outbound FOSS contributions are another way to give back to the community. Supporting and donating to free and open source projects proves the company is a responsible corporate FOSS citizen, not counted among the many organizations that siphon off the work of volunteers without so much as a word of gratitude.

For instance, for several years the software monitoring platform company Sentry has contributed increasing amounts of funding directly to free and open source software maintainers.⁴ As of 2023, they were contributing some sort of funding to nearly 100 percent of their FOSS dependencies. Sentry isn't doing this for the branding; instead, the goal is to sustain their FOSS software supply chain. Their improved brand and a growing reputation for sincerely and respectfully supporting FOSS maintainers is, however, some lovely icing on an already delicious cake.

Recruiting, Hiring, and Retention

Actively engaging in FOSS communities via outbound contributions can be great for staffing. By earning the reputation of being a good FOSS citizen, your company also becomes an enticing place to work. More than ever, people

3. <https://corpgov.law.harvard.edu/2011/08/20/making-the-business-case-for-corporate-philanthropy/>

4. <https://blog.sentry.io/we-just-gave-500-000-dollars-to-open-source-maintainers/>

are using that reputation when deciding whether to work for your company.⁵ When competing to recruit for roles like programmer, product manager, or others where you may face a lot of hiring competition, having the brand of FOSS citizenship can give your organization the edge it needs to bring the best people on board.

Contributing to and participating in FOSS communities gives your hiring a boost in other ways as well. For instance, because all work on free and open source projects happens out in the open, your organization has a front-row seat to all of the skills on display. Do your teams need people with good communication or project management experience? Intimate knowledge of a particular tech stack? When you contribute to a project, you can see exactly who can do what you need as well as just how good they are at it, then approach them about joining your team. It's a lot easier to find those qualified candidates when you and they are all hanging out in the same place, after all.

Another benefit of hiring from the FOSS communities in which your company participates is that these people are more likely to be familiar with the technology used inside your company. That increases new hire comfort levels, decreasing onboarding time and making everyone more productive more quickly.⁶

At the other end of the staffing timeline, outbound FOSS engagement improves staff retention. It provides superb opportunities for professional development for roles across the entire organization, not only for the programmers. By supporting employee participation in free and open source software projects, your company is strengthening the skills of not only its team members but also the entire ecosystem of FOSS on which its products rely.

Market Research and Business Information

Have you ever gone to a professional conference or event? You probably learned a lot from the scheduled presentations but noticed that you got valuable information of a different sort from the informal gatherings that happen in between those sessions. This *hallway track*—so named because the interactions often occur in the hallway between rooms—provides priceless networking opportunities as well as the sort of information sharing you'd never get from a PowerPoint slide deck.

5. <https://hbr.org/2022/02/make-your-employer-brand-stand-out-in-the-talent-marketplace>

6. <https://ibima.org/accepted-paper/impact-open-source-recruitment-case-digital-companies/>

FOSS communities are the never-ending hallway track of the software world. With outbound FOSS contributions and participation, your company can learn so much more than simply how to use and maintain a FOSS component:

- You can learn how *other people* use the component and whether their needs are met by it.
- If your company has a product offering built around the component, this sort of market research is invaluable.
- Keeping your ears and eyes open in this hallway track can also tip you off to the Next Big Thing before it hits the mainstream, giving your company a first-mover advantage over your competitors who aren't contributing to FOSS.

For example, if your company is in the automotive industry, it may wish to attend and participate in events staged by the Eclipse Software Defined Vehicle open working group.⁷ By doing so, not only do you get to shape the direction of the open source standards and software stacks used in future vehicle platforms, but you'll also learn a lot about what's happening in the industry and what your end users will want in the final product. This sort of knowledge is priceless when developing the long-term product plans required in an industry like this.

How Releasing FOSS Can Benefit Your Strategy

Where outbound FOSS engagement is concerned, contributing to and supporting projects is a good way to get started. After all, contributing to something that already exists is a lot easier than creating and releasing something new. It's also a lot wiser to improve the wheel you have rather than reinvent it. Releasing FOSS is more of a challenge, but it's still worth the effort.

You'll learn a lot more about releasing projects under free and open licenses in [Chapter 19, The Strategy of Releasing FOSS, on page 297](#), and [Chapter 20, Releasing FOSS Projects, on page 317](#), but to get you thinking, here's a brief summary of some of the ways this can help you reach your business goals.

Share the Maintenance Burden

Frequently, your teams need to build tools, utilities, and other resources to solve problems that arise in the course of their jobs. These tools aren't intended to become a part of any product offering and aren't differentiators for your business but are helpful and make lives better for your employees.

7. <https://sdv.eclipse.org>

Maybe it's a script to convert a lot of files from one image format to another, or perhaps it's a collection of stock photography from the regions where your company does business.

Whatever it is, even if someone created it as a one-off (a “hack”), its usefulness means it not only sticks around but also needs updates and maintenance. What started as a quick hack has now become a burden on those who need to maintain it.

In cases like this, it can be helpful to release the tool under a free and open license so others can use it to solve their own problems as well. Your company gains the reputation boost from sharing with others, and it also receives contributions and maintenance assistance from the wider community.

Netflix releases many of its internally developed utilities for others to use and contribute to.⁸ For example, the company created a suite of tools it calls the Simian Army.⁹ Featuring the Chaos Monkey¹⁰ project, these tools allow Netflix to test the resilience of its services to ensure a good experience for all its subscribers.

Netflix isn't in the business of *chaos engineering*; it's a streaming entertainment service provider. Releasing its Simian Army tools provides a great deal of benefit to the overall technology industry but doesn't negatively affect Netflix's business model. Instead, by sharing the projects, Netflix receives contributions and input from people with a more diverse set of use cases, thereby improving the software for everyone.

Many hands make light work, but it's worth noting that many hands don't *eliminate* the work required. As you'll learn in [Chapter 20, Releasing FOSS Projects, on page 317](#), releasing projects takes effort, both up-front and later for things like community management.

Gain Collaborators

When your company releases a FOSS project, it's essentially hanging out a “Help Wanted” sign to the world. Or maybe the sign reads “Free Tomatoes.” Or even “Free Puppies,” if you hope that someone else will be taking care of them. Regardless, the expectation is that other people and organizations will come by, have a look, and maybe use and contribute to the project. Anywhere two or more people get together to work on something, you'll find collaboration.

8. <https://netflix.github.io>

9. <https://github.com/Netflix/SimianArmy>

10. <https://github.com/netflix/chaosmonkey>

Of course, collaboration doesn't always go the way you expect. Disagreements in the direction to take the project may come up, or the project itself may be poorly managed, so the collaborators aren't able to make much progress. These are solvable problems—even avoidable ones—and shouldn't scare your company away from releasing FOSS projects.

Nor should any perceived threat that a competitor may come along to use and contribute to the project scare you off. This *coopetition* can lead to great results for the project when it solves problems for all of the stakeholders. The proof is evident in nearly every nonprofit FOSS foundation.¹¹ The Open Infra Foundation, the Eclipse Foundation, the Apache Foundation, and many more show the power of corporate members coming together to collaboratively develop the software on which they all rely.

Always keep in mind where your business actually competes; odds are it's not differentiating its products based on things it releases under FOSS licenses. You'll learn more about this in [Chapter 19, The Strategy of Releasing FOSS, on page 297](#).

Use It as a Loss Leader

Maybe your business *has* released a differentiator as a FOSS project. As you'll learn in [Chapter 22, Incorporate FOSS into Business Models, on page 369](#), this can be an element of a *freemium* (also known in software as *open core*) business model. By releasing the core of a product offering under a free and open source license, the company can use it as a loss leader, attracting people to the free product to tempt them to purchase additional functionality, support contracts, or other product offerings.

Font Awesome provides high-quality icons and a toolkit to speed the design and development of websites and software applications.¹² Many of their designs require a subscription for use but not all of them. Font Awesome's open source icons have become a keystone of web and app design. By providing great-looking icons to anyone who needs them, Font Awesome not only makes the web a lot more pleasant to look at and use, it also raises awareness of its products and services in millions of people. You'll learn about this and other strategic reasons to release FOSS projects in [Chapter 19, The Strategy of Releasing FOSS, on page 297](#).

11. <https://flossfoundations.org/foundation-directory/>

12. <https://fontawesome.com>

Own an Ecosystem or Market

You've done all your research. You've listened to potential customers. You've found the perfect product-market fit. Now how can you get the edge in the market, limiting the opportunities for your competitors? Release your product under a free or open source license.

Properly managed, a FOSS product offering develops a following and a community. Customers cultivate an affinity for the product brand, leading to a set of marketplace expectations which all potential competitors must meet if they wish any chance of success. Many organizations may choose not to compete at all rather than play the game on your terms. It can be hard to pull off this sort of business move, but if you can do it, then you'll find it's a powerful one.

Drive an Emerging Standard

Growth-focused companies understand that without innovation they'll never meet their ambitious goals. People in these companies are always on the lookout for the Next Big Thing (NBT). More often lately, that NBT is built around FOSS and/or open standards.

By participating in these NBT projects, your company can get the edge in the competitive marketplace. Not only do you get to shape the NBT as it's being created, you also gain valuable insight into NBT trajectory and schedule. If you're building a product offering around this NBT, this information is priceless for your product development. Releasing compelling products and FOSS projects built around an emerging technology or standard drives adoption and cements your company's strategic decision to support the standard itself.

The RISC-V (pronounced *risk five*)¹³ instruction set architecture (ISA) is an example of the power of engaging with open projects and standards. This ISA, stewarded by the RISC-V International nonprofit organization, is openly licensed under the Creative Commons Attribution 4.0 International license,¹⁴ allowing anyone to use and implement the standard without any license fees typically found with ISA specifications. While it's developed in the open and anyone may comment on or contribute to the specification,¹⁵ only members of RISC-V International may vote on what gets included in the specification.

13. <https://riscv.org>

14. <https://creativecommons.org/licenses/by/4.0/>

15. <https://github.com/riscv/riscv-isa-manual>

Though still fairly young as ISAs go, RISC-V is already proving to be successful. Even people who don't work in the CPU or chip-making industries will recognize many of the names of companies that either are or already have developed and released CPUs based on the RISC-V architecture: Google, Nvidia, Fraunhofer, and others. Many of the participating companies have been RISC-V International members from nearly the beginning, allowing them to not only shape the direction of the specification but also develop and release their chips before their competitors. In turn, these chip releases drive more interest in the architecture and continually increase the market for it.

Build Ecosystem Beyond the Product

Even if you don't release a product offering under a FOSS license, you'll still find strategic reasons to take advantage of an open ecosystem.

For instance, you can support the creation of third-party add-ons and modifications to the product. Some companies do this by providing openly available interfaces to the product. Anyone can develop a new addition to the product, but only customers who buy the product can benefit from it. Enabling the creation of this sort of third-party ecosystem around your product offering makes the product more enticing to new customers and makes existing customers less likely to leave—which is to say, it makes the product more *sticky*.

Video game developers do this all the time by embracing and supporting “modding” communities, where players come together to create and share their modifications to the game. The farming simulation game *Stardew Valley* is rapidly approaching its tenth birthday, yet it remains popular both because of the game itself and because its author embraces the abundant ecosystem of fan-created modifications. This gives the game life above and beyond its original purpose and ensures people will keep buying and playing it far longer than the lifespan of your average game.

Agile and User-Centered Product Development

In [Innovation and Digital Transformation, on page 87](#), you learned about how FOSS engagement can help with your organization's digital transformation effort. For many organizations, that transformation includes transitioning toward more user-centric product offerings. Often this process involves adopting *agile development*.

The Agile Manifesto—the document that led to the agile product development movement—has four principles.¹⁶ Releasing FOSS projects helps to support each of them.

Individuals and Interactions over Processes and Tools

Looking at the current state of agile software development today, it's astonishing how many people ignore the first principle entirely. Instead of focusing on the end result—how well the software addresses user needs—software development teams become buried in meetings, metrics, charts, process overhead, and tooling to support it all. No doubt about it, “agile” has become a big business for a lot of tooling development companies.

Processes and tools are vital elements of any software development process, of course, but the important part is how those elements support creating value through software, not the other way around. Releasing software as FOSS is like standing in the agora in ancient Athens. It invites conversations from interested parties, informing how the matter takes shape. This happens despite the processes and tools, not because of them.

Working Software over Comprehensive Documentation

This is the most misunderstood of the agile principles. Rather than meaning “no documentation needed,” as people regularly interpret it, it actually means that the sooner you can put a working example in front of someone, the sooner you can receive and then take action on feedback about it. The alternative at the time of the Manifesto publication was to spend weeks, months, or even years writing detailed design documents and specifications. This is the “comprehensive documentation” referred to by the principle, not documentation to support using and contributing to the software.

FOSS projects provide the most opportunities to receive feedback by maximizing the number of people who are able to try out the software across its entire life cycle as well as by having established best practices around providing that feedback.

Customer Collaboration over Contract Negotiation

In more traditional enterprise software acquisition cycles, a vendor salesperson has frequent calls with executives in the software-acquiring company to negotiate what features they'll include. This may require custom development on the part of the vendor. Similarly, internal software

16. <http://agilemanifesto.org>

tooling development teams can spend weeks in meetings with internal stakeholders, haggling over every conceivable detail for a new tool. Neither of these processes results in satisfactory software for the end users, and some sort of renegotiation is always required.

Compare this to the agile practice of presenting the end user with a Minimal Viable Product (MVP), collecting feedback, and then iterating from there to develop precisely the software required. Shipping that MVP as a FOSS project or using FOSS development best practices makes the feedback and iteration process easier and more streamlined for everyone.

Responding to Change over Following a Plan

This principle sums up the spirit of the entire Manifesto. True agility is being able to respond to changes in the environment, ecosystem, marketplace, customer requirements, or in whatever system you're trying to operate your business. Plans are, of course, necessary for setting direction, but having the ability to change the plan as the environment evolves is the superpower of a truly agile organization.

Each one of these principles applies and is critical to releasing and maintaining a successful FOSS product. The interactions enabled by free and open source communities lead to immediate feedback, short product development cycles, and products that provide immediate value.

This sort of user-centered product development method, when used as a part of a larger strategy, can be powerful in a competitive marketplace.

Risks of Outbound

Of course, benefits also come with risks, and outbound FOSS is no different. However, as with most risks, those involved with outbound FOSS are avoidable if you're looking out for them.

Unfortunately, fear of these risks often prevents organizations from engaging with FOSS at all, despite benefitting from the work of external volunteer free and open source software communities. Advance knowledge of these potential risks can help you not only to avoid them but also to alleviate any anxiety and misgivings that your leadership may have about the endeavor. When they see you've already planned for all the contingencies, they're more likely to green-light the effort.

Here are the most common risks people experience or worry about when starting an outbound FOSS initiative.

IP Exposure

The first risk most leaders think about when faced with the idea of releasing or contributing to free and open source software projects is losing control of their organization's precious intellectual property. "Oh no!" they think, "Someone will see how we do things here and then we'll be ruined! Alas and alack!" OK, perhaps they're not so dramatic about it, but they're certainly worried that outbound FOSS will unnecessarily expose company IP. The risk of exposing something that will harm the business is real, but it's not as likely as many would have you think.

If you recall from [Copyright, on page 24](#), original expressions of ideas are intellectual property and more or less automatically receive copyright protections. This is regardless of whether that idea expression is a trade secret, a patentable business process, or a memo detailing that employees should not microwave fish in the shared staff kitchen.

I'm sorry to be the one to have to break it to you, but most of the intellectual property that your company creates isn't especially important, and it's certainly not business critical. Across your organization, people create stacks of copyrightable works every week, and most of those are of the fish-memo sort: works that improve life for those who make the business tick but don't directly impact the business itself. Utilities, templates, and similar things usually fall into this fish-memo category and often are safe candidates for releasing under free and open licenses. Contributions to FOSS components that the team uses to build the company's web presence are similarly safe; these are an example of an outbound engagement that's unlikely to expose business-critical intellectual property.

What qualifies as critical intellectual property naturally varies by company and business, but you'll probably know it when you see it. Does the IP directly touch upon business matters? It's likely critical. For instance, if you're a car manufacturer, then details about the internal workings of the infotainment system in your next-gen electric vehicle may be differentiating functionality that you don't want exposed to competitors.

Developing, communicating, automating, and enforcing FOSS policies and procedures (P&P) can work wonders for mitigating this sort of outbound engagement risk. Revisit [Chapter 10, Get Started with Your Inbound Policies, on page 163](#), and [Chapter 9, Establish Effective Inbound Policies and Procedures, on page 145](#), for more information about creating P&P that work. And never forget, where IP is concerned, if there's ever even a hint of a question,

consult your legal team. They'll let you know whether that outbound engagement is a legitimate risk to the business or whether it's just a fish memo.

Reputation

Another commonly raised concern is that the outbound contributions won't be high quality enough and will reflect poorly on the company.

To avoid any potential damage to their reputation, the company leaders either ban any outbound FOSS engagement or they create such burdensome outbound P&P that the end result is effectively the same as a ban. This is the FOSS equivalent of not inviting your friends over to your flat because you always forget to dust or because the sofa cushions don't match the curtains. The only one who cares about those cushions is you. Your friends are just grateful to spend time with you.

That isn't to say outbound contributions and project releases shouldn't have review before they leave the company; they absolutely should! We all have blind spots, and getting a second, third, or umpteenth opinion can help our work improve. That's why many FOSS projects have a multistage review process for contributions.

When developing your outbound initiative, come up with a realistic list of things for which you'd like to check before outbound contributions and projects leave the company, the hidden landmines that could cause legitimate business pain, then create P&P that will ensure those landmines are found and defused before they have the chance to hurt anyone. This list of landmines will differ based upon your organization and business risk, but these are a few items that typically end up on it:

- Does the project or contribution work at all?
- Is there adequate documentation, and does it make sense?
- If it's software, are there installation and usage instructions? How's the code coverage for the tests?
- Are there profanities or other bits of questionable language in there, including negative comments about competitors or other brands?
- No one is sharing sensitive information like login credentials or customer data, are they?
- Is there anything patentable in there, and if so, are we a company that cares about that sort of thing?

The importance of each item on the list is relative to your business needs and risks, but many of them lend themselves to a “good enough” review and revision rather than requiring perfection. Remember, your friends don’t care if the pillows don’t match the curtains, but if it’s important to you, then you can have your friends over and then get around to fixing the cushions when you have more time—or, if it’s FOSS, someone else can get around to it for you.

Iteration and collaboration are the name of the game, so release those outbound contributions early, release them often, and fix them afterward as you learn more about end-user needs.

Competition

When faced with the possibility of contributing to or releasing FOSS projects, some people will object. A common misconception is that any outbound FOSS engagement will be a valuable business resource for their *competitors*. Any FOSS contribution or project your company releases would be like giving a gift to the competition, providing them insights and tools that will allow them to get ahead in the marketplace at your company’s expense.

Always remember the business that you’re in, the business model your company uses, the differentiating factors that allow it to compete successfully, and the business goals you’re targeting with your overall strategy. You’ll learn more about this in [Chapter 19, The Strategy of Releasing FOSS, on page 297](#).

Now, with those in mind, review the plan for outbound FOSS engagement. Will a plan for contributing internally developed functionality to, for example, the pandas Python data analysis library, releasing custom-designed emoji, or sponsoring the maintainers of load-bearing FOSS components benefit your competitors more than your own company? If so, that plan needs to go back to the drawing board.

If you’ve developed the plan strategically, then you’ll be able to point to relevant business elements and show how the plan moves the company toward its goals. Importantly, you’ll also be able to communicate how the outbound engagement takes your competition into account.

Government Regulations

Government regulations are an important part of any business undertaking, and outbound FOSS is no exception. Thankfully, the “default to open” nature of free and open source can make regulations a little easier to manage for outbound, but that doesn’t mean you can just ignore them.

Most of the regulations and government orders that may touch upon FOSS relate to consumer protections and cybersecurity. The following are some examples:

- European Union Cyber Resilience Act¹⁷
- European Union General Data Protection Regulation¹⁸
- United States Executive Order on Improving the Nation's Cybersecurity¹⁹
- India Digital Personal Data Protection Act, 2023²⁰
- India National Cyber Security Strategy 2020²¹

As you can see from the titles of these documents, none of them are specific to open source software but instead apply much more broadly. This means your company may already be complying with any relevant regulations in the course of its normal business operations, which is good news for outbound FOSS initiatives. Your organization will have the infrastructure and processes in place to ensure regulatory compliance, so you'll probably be able to incorporate that existing work in your outbound FOSS P&P.

Similarly, export controls are another type of government regulation that *may* apply to your outbound FOSS initiatives but also may not, thanks to the FOSS cultural requirement to do all work out in the open where everyone can see it. In the United States, your outbound FOSS initiatives are unlikely to trigger any export controls unless they include encryption,²² but things may get more complicated in other jurisdictions.²³ Relatedly, if the FOSS project is based in a jurisdiction that has placed sanctions on other countries, it may impact contributions, as people from sanctioned countries may not be allowed to contribute to the project.

Regardless, IANAL and TINLA (remember these from [Lawyers Are Your Friends, on page 18?](#)): I am not a lawyer, and this is not legal advice. Where this and any other government regulation are concerned, always stop and consult qualified legal specialists before taking action.

17. https://en.wikipedia.org/wiki/Cyber_Resilience_Act

18. https://en.wikipedia.org/wiki/General_Data_Protection_Regulation

19. <https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/>

20. https://en.wikipedia.org/wiki/Digital_Personal_Data_Protection_Act%2C_2023

21. <https://database.cyberpolycyportal.org/api/files/1664377398590fe3kyzt8xwd.pdf>

22. <https://www.linuxfoundation.org/resources/publications/understanding-us-export-controls-with-open-source-projects>

23. <https://academic.oup.com/book/44727/chapter/378967490>

Information Security

As you likely noticed in the preceding section, a lot of the government regulations listed are related to privacy and cybersecurity. These things are a big deal, but you don't need the government to tell you that. Your company already (hopefully) takes privacy and security seriously, making sure all critical business resources are locked up behind various firewalls and authorizations.

You likely also already know that privacy and security isn't a set-it-and-forget-it thing. Criminals are always evolving their nefarious ways, so your company's information security (infosec) team similarly needs to stay on its toes to keep the bad guys out of the good stuff.

With all this attention on keeping data and trade secrets safe inside your company, beware you don't waste all that effort and make it easy for criminals by inadvertently publishing sensitive information with your outbound initiatives.

For instance, many companies have learned the hard way that it's a poor practice to keep their cloud computing credentials in a version control system, and they usually learn this when they release those credentials as a part of an outbound FOSS initiative. Once these credentials are released as a part of a FOSS project, it's already too late.

Remember Craig Hays' experiment from [Security Considerations, on page 77?](#) It only took 34 minutes for someone to locate credentials Hays had placed on GitHub, then use those credentials to gain complete access to the server. The best a company can do at that point is rapidly change the credentials and hope that they discovered their mistake before anyone was able to take advantage of it.

This sort of thing definitely is a risk, and one that can turn into a nightmare scenario, but it's also easily avoidable. Like most security matters, it involves applying a little common sense.

First of all, make sure to follow best practices for securely developing any software or materials your company creates. Your own infosec team will be able to provide guidance here, but if you don't have a team for this, then organizations like NIST,²⁴ SAFECODE,²⁵ and the OpenSSF Best Practices Working Group²⁶ provide resources that will help.

24. <https://csrc.nist.gov/Projects/ssdf>

25. <https://safecode.org/category/resource-secure-development-practices/>

26. <https://github.com/ossf/wg-best-practices-os-developers>

Then, when you're ready to contribute or release materials for outbound FOSS, make sure the P&P you learned about in [Reputation, on page 257](#), include scanning for sensitive information prior to any contributed stuff making it out of the company and into the public eye.

Free as in Puppy

Would that this were a risk more companies were aware of. Let's fix that now!

Remember *free as in puppy* from [Maintenance Costs, on page 83](#)? It's the situation where you acquire a loveable new burden on your company's time and energies, much like bringing a puppy into your life. When your company engages in outbound FOSS, it has metaphorically acquired a puppy and has taken on responsibility for its care and feeding.

You'll learn the details in [Chapter 20, Releasing FOSS Projects, on page 317](#), but basically when your company releases or supports a FOSS project or community, it tacitly agrees it will help to keep it healthy. This doesn't only mean responding to feature requests and fixing bugs, it also means managing the community to ensure it's welcoming and functional.

Far too many companies mistakenly see outbound FOSS engagement as a one-time transactional thing: we, the company, give this contribution or release to you, the public, and in return you will adore us. OK, perhaps "adore" is over-selling it a bit, but the transactional assumption holds true. It can be OK for your organization to have casual interactions with FOSS projects like this. Sometimes noncritical projects require only minimal attention from your company. Critical projects, however, are like a puppy, requiring attention durations that are numbered in years rather than months or days. For the load-bearing FOSS projects of your business, transactional, casual interactions rarely provide the strategic support required for your company's needs.

As you develop your outbound FOSS strategy, take the long view with regard to any efforts you're planning. Don't be surprised by the responsibility of maintaining FOSS projects through their entire life cycle. If you need (for budgetary, strategic, or other reasons), include checkpoints and graceful exit strategies in your plans. If you don't, and your company abruptly withdraws from a FOSS project that it has released or that has come to rely upon it, it's likely to do serious damage to the brand of your business.

Now that you have an overview of how outbound FOSS can benefit (or hurt) your company, it's time to put that knowledge to work. Our first stop: contributing to existing FOSS projects.

Master Contribution Fundamentals

Contributing to free and open source projects is such a large topic that I've written an entire book about it.¹ That work covers the culture and mechanics of how an *individual* can make FOSS contributions, so it'll be a valuable resource for your team members who are tasked with engaging with FOSS communities. The bigger picture of contributing from an organizational perspective is a bit different, though. In this chapter you'll learn the fundamentals for contributions from a business point of view.

Contributions Sustain FOSS and Your Business Both

You've already learned in [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#), and [Chapter 13, Sustain. Maintain. Retain., on page 213](#), why supporting FOSS can be a good business investment, but here's a brief recap of some of the ways that FOSS contributions are a valuable addition to your strategy toolbox.

Every supply chain has some links that if they broke would throw a business into disarray. These *load-bearing links* (see [Identify and Evaluate Critical Links in Your FOSS Supply Chain, on page 219](#)) exist in your software supply chain (SSC) as well and often are FOSS components. By contributing to the projects and communities that support these links, your company helps to ensure their strength and sustainability, bringing with it the peace of mind of knowing your FOSS supply chain can support your business without any interruptions.

In the last chapter you learned that FOSS contributions can help to strengthen brands (see [Reputation/Brand, on page 247](#)). Giving back to the community is one way to build your company's brand, and FOSS contributions

1. <https://fossforge.com>

are a good way to do that. Individual volunteers will always be the foundation of free and open source software communities, but strengthening that foundation through corporate assistance proves your company understands its responsibilities as a citizen of the FOSS ecosystem.

That citizenship is something that all in the ecosystem should take seriously. Companies that take from the ecosystem without giving back to it can look like parasites, exploiting the work of volunteers to line their own pockets. A parasitical or freeloading reputation can make it more difficult for an organization to recruit and retain people for certain roles, especially those in highly competitive technical areas. Remember, from [Maintenance Costs, on page 83](#), that FOSS is *free as in puppy*; like a puppy, it requires care and attention to thrive. Similarly, those who abuse it are looked upon with scorn. Contributing guarantees your company will gain the reputation of responsibility by helping to raise the puppy.

Ways to Contribute

When people hear the phrase “contribute to FOSS,” their minds usually jump immediately to programming and contributing code. That’s certainly one way to contribute, but innumerable others are possible. In fact, for some projects code contributions aren’t even the best way for your company to make a positive difference. Once when I was speaking with Tiffany Farriss of the Drupal Association,² she provided the perfect summary of the different ways companies and individuals can give back to FOSS communities: time, talent, and treasure.

Time

Any sort of contribution your company decides to make will require that it allows people to spend company time on it. Time is actually the most precious contribution anyone can make to a cause. Once it’s spent, you never get any more, so you should always be sure you—and your organization—are spending it wisely.

If FOSS engagement is a part of your company’s strategy, plan for this time usage in advance. Make the process as streamlined as possible; find ways to clear the path for those who’ll be contributing so they don’t waste that time jumping through administrative hoops.

2. <https://www.drupal.org/association/board>

One way to do that is to create an approve list of critical, load-bearing FOSS projects for which any type of contribution is allowed on company time. Having this list resolves many potential problems up front:

- Your company can sign any necessary CLA so there'll be no delays or misunderstandings to deal with (refer to [Contributor License Agreements, on page 34](#), for more information).
- Your Open Source Program Office (OSPO) can coordinate with the projects to determine how best to meet their needs.
- Your employees have a ready-made list of project options, making it easier to get started with contributing.

Commit to Contributing

All too often I've seen companies pay lip service to supporting staff members contributing to FOSS projects and then not allow them the time to do so.

I understand that a lot of things need to get done in a business and that some tasks naturally will be a higher priority than others. However, if company leadership says one thing but their actions prove another, it only serves to undermine employee trust and pollute the culture from the top down.

If outbound FOSS engagement is important enough to become a part of your company's strategy, then it's important enough to carve out the time to allow it to happen. Should you find that people keep getting pulled away from their FOSS contributions to work on other company tasks, the problem is unlikely to be that you've set aside too much time for contributions. It's much more likely that the underlying problem is in time management or in understanding or communicating priorities.

Don't give up on outbound FOSS engagement. Address any underlying (and admittedly more difficult) prioritization and/or political problems first before deciding that FOSS contributions aren't going to work for your company.

Talent

As skilled as we all are, none of us can do everything. Certainly the limitation of time is one reason for this but also a lack of knowledge and experience. This is why your company employs specialists, such as information architects, accessibility designers, and facilities managers.

Free and open source software is just that: software. All successful software projects require skills aside from programming: marketing, design, security,

IT, writing, and more. Your company is in the position to provide the talent that FOSS projects need to thrive.

Non-code contributions are a critical part of any FOSS project and often are the sorts of tasks that the FOSS community itself is lacking the time or skills to manage. Non-code contributions can include these:

- Helping to organize and run events
- Managing the blog, social networks, and other marketing outreach
- Triaging tickets/issues
- Project management
- Community management
- Updating the website
- Running security audits

By not limiting itself to code contributions, your company can get creative with its contribution strategies and include more of the company in them. However, before you flex those creative brain cells, start by having your OSPO approach load-bearing projects to ask where they need assistance. This way you can make sure your company's contributions really make a difference to the projects.

Treasure

Sometimes the answer to the question “how can we contribute?” is “throw money at the problem.” These monetary contributions can be a part of a larger outbound FOSS engagement effort, or perhaps they're the entirety of that effort. Maybe your company is just getting started with its outbound strategy and has yet to ramp up the necessary internal processes and communication paths, and providing financial support is a way you can start making a difference quickly—but hopefully not so quickly as to be reckless.

As you learned in [Monetarily Sponsor Projects, on page 224](#), financial support can come with its own complications for the FOSS maintainers receiving the funds. They may not be prepared to receive money or may not know how to manage it responsibly with regard to taxes and the like. Despite that, don't shy away from the option to make monetary or in-kind contributions to FOSS projects. These can allow members of the FOSS community to dedicate more time to the project, scale up overloaded infrastructure, or even hire assistance dedicated to adding much-requested features.

Pay the Maintainers

In [The Situation with FOSS Sustainability, on page 217](#), you learned about the *Roads and Bridges* report by Nadia Asparouhova. Ever since the Ford Foundation released the report in 2016, there's been a steady call to action coming from the FOSS ecosystem: “pay the maintainers.” In answer to that call, schemes to get money into the hands of FOSS maintainers have been popping up like mushrooms after the rain. Some of them—such as PayPal, Ko-fi, and Patreon—aren't specific to the FOSS maintainer use case. Others—such as GitHub Sponsors³ and OpenCollective⁴—launched specifically to address the question of how to get money into the hands of FOSS maintainers. The functionality and currency support of each differs, but the end result is the same: money finds its way to the FOSS project maintainers.

Fiscal Sponsorship

Some projects have chosen to join *fiscal sponsors*. By doing this, the FOSS projects gain the services and support offered by the sponsor, and they also get to benefit from the sponsor's not-for-profit status. Software Freedom Conservancy (SFC or just Conservancy)⁵ and Software in the Public Interest (SPI)⁶ are two of the nonprofits that operate as fiscal sponsors for Free Software projects.

Contributing to projects that have fiscal sponsorship can be beneficial to all parties. Depending upon your jurisdiction, donations to the project by way of their nonprofit sponsor may qualify as tax deductions for your company. For that reason, among others of standard business practice, always make sure to consult with your financial team before starting any sort of monetary support program for FOSS communities.

Create a Corporate FOSS Maintenance Fund

An increasing number of companies are starting to take to heart the call to pay the maintainers of the FOSS projects on which they rely. They're answering that call by creating FOSS Contributor Funds.

3. <https://github.com/sponsors>

4. <https://opencollective.com>

5. <https://sfconservancy.org>

6. <https://www.spi-inc.org>

The implementation of these funds varies by company but often includes these elements:

- Set aside a regular and recurring budget earmarked for FOSS projects.
- Determine criteria for which projects may qualify for funding. Criteria can include project characteristics such as the project is a load-bearing link in the company's free and open source software supply chain (SSC), the project has a well-defined and documented method for accepting funding, and the project has not received funding in a certain period of time (past quarter, year, and so on).
- Analyze the FOSS supply chain and create a list of qualified projects based upon the predetermined criteria.
- Determine a general formula for calculating how much funding should go to each selected project. Will every project receive the same amount? Will you vary it by project size or by criticality to your company's business operations? Only you will know how best to judge what's right for your company and its unique constraints.
- Select which projects from that list will receive funding in this round. Some companies allow staff members votes to decide this matter.
- Send out the money and make FOSS maintainers happy.

In practice, this process is complex and can take a lot of time and thought to put together and implement. Thankfully, some very smart and generous people have created resources to get you started.

At the forefront of those generous souls is Duane O'Brien, a public advocate for and specialist in corporate funding for FOSS projects. He and his team wrote and published *Investing in Open Source: The FOSS Contributor Fund*,⁷ releasing under the Creative Commons Attribution-Non-Commercial-NoDerivatives 4.0 International License.

Your company can also join FOSS Funders,⁸ a community of practice, for mutual support and assistance.

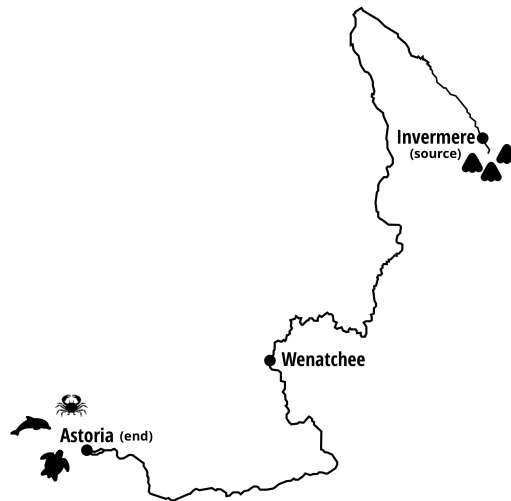
7. https://github.com/indeedeng/FOSS-Contributor-Fund/blob/main/Investing_in_Open_Source-FOSS_Contributor_Fund.pdf

8. <https://fossfunders.com>

Downstream vs. Upstream

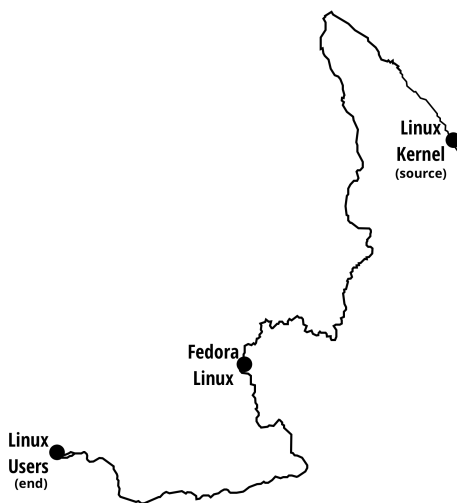
As you work with and in FOSS projects and communities, you'll regularly hear people referring to "upstream" and "downstream" open source development, but it's rare that you'll find anyone explaining it. It's usually taken as a given that you know where you are in relation to this "stream" everyone's talking about, and that can be confusing.

To explain it, let's take that metaphorical stream and describe it in terms of a real one: the Columbia River. OK, so it's unfair to call the fourth largest river in North America a stream, but the Columbia is secure enough in its identity that it won't mind. The river starts in British Columbia, Canada, in the Rocky Mountains near the town of Invermere. It then continues through the Pacific Northwest of North America for more than 1,200 miles (2,000 kilometers) to crash into the Pacific Ocean via the deadly Columbia Bar in Astoria, Oregon, in the United States. See the following figure:



If you know anything about rivers, you know that regardless of where you are on the river its source is always upstream and its mouth downstream. The water flows from the source in the high mountains (Invermere) down to the mouth at the ocean (Astoria). Astoria is downstream of Wenatchee and Invermere; Wenatchee is upstream of Astoria and downstream of Invermere, and Invermere is upstream of all of those. Everything that's downstream depends upon the existence of upstream in some way, but things upstream can exist without the downstream. If there were no Astoria, Invermere likely would not notice (sorry, Astoria).

Linux distributions (*distros*)⁹ are a good example of applying this concept to FOSS. Linux has dozens of distros, some more popular than others. In this example we'll use Fedora Linux. The Linux kernel forms the basis of Fedora and therefore is upstream of the distro. Any users of the Fedora operating system are downstream of both Fedora and of the Linux kernel. If there were no kernel (upstream), there'd be no Fedora Linux (downstream of the kernel) and therefore also no users (downstream of Fedora), but if there were no Fedora, there could still be a Linux kernel. The following figure illustrates:



You've seen the example here with a river, but it also works with other concepts. In your business, your company is downstream from its suppliers but upstream from its customers. My tea is downstream from the tea company, which is downstream from the tea growers.

Looking at it this way, the whole upstream/downstream thing is simply another way to think about supply chains. Until recently though, the concept of *supply chain* wasn't common in the FOSS ecosystem, and so the upstream/downstream metaphor prevails. Keep this in mind as you engage with FOSS projects in your outbound initiatives.

Upstream Development

Contributing directly to a FOSS project is *upstream development* and is the default mode for most contributions to free and open source software projects. These contributions can be any of the types you learned about in [Ways to](#)

9. https://en.wikipedia.org/wiki/Linux_distribution

[Contribute, on page 264](#), but when someone uses the phrase “upstream development” they’re usually talking about software source code contributions.

“Upstream” doesn’t necessarily mean contributing all the way up to the source of the stream itself. Someone can send a patch to the Fedora distro for any of the other components of that operating system, without necessarily having to contribute to the kernel at the source of that particular stream. Remember: “upstream” and “downstream” are relative to where in the stream you’re standing.

Downstream Development

While it’s most common to contribute to the upstream FOSS project, it’s possible to make and maintain any necessary changes where you are, downstream from the source. This is called making an *internal fork* of the project.

If you recall from [Do You Have the Spoons to Deal with a Fork?, on page 45](#), when you fork a FOSS project you’re making a copy and rereleasing it under a new name and with a new community. Internal forks are similar, but the project doesn’t get released publicly. Instead, your organization maintains the copy internally, making any necessary changes on the copy rather than contributing them upstream.

Here are some examples of why your company may decide to maintain an internal fork:

- A newer version of the FOSS component has announced it has fixed a security vulnerability, but your company isn’t yet in a position to upgrade to the new, secure version of the software. Instead, you download the source of the version that’s currently in use in your products, apply the security fix to this older version of the software, and maintain that internal fork until you’re ready to upgrade the FOSS component.
- The commonly distributed upstream project doesn’t have the configuration your company needs, so you download the source from upstream, make the necessary configuration changes, and use the locally created software build in your product.
- Your company has decided to build a unique product offering around an existing FOSS project.

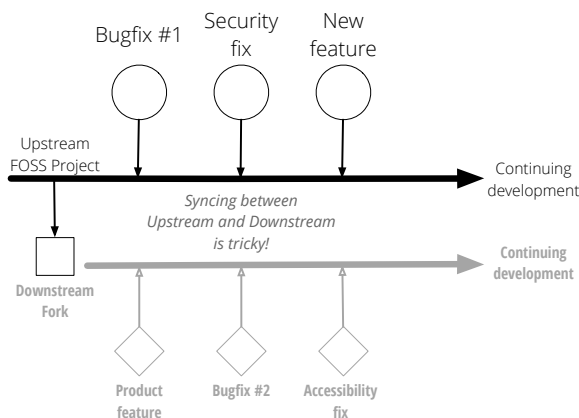
You might fork a FOSS project for downstream internal development for any number of other reasons, but all of them should be very carefully considered prior to taking action. While the initial acquisition of the project source code and related materials may cost your company very little,

ongoing maintenance of it can be expensive. Instead, it's usually best for your company to operate according to an Upstream-First Policy.

Upstream-First Policy

Maintaining an internal fork is an expensive proposition. Not only does the team need to maintain their own changes to the FOSS project, they also need to keep it in sync with developments happening in the upstream project.

Let's say your company forks a FOSS project and, for whatever reason, decides to maintain it as an internal fork. Time passes, because that's what time does. Meanwhile, the upstream FOSS project continues on its merry way, adding bug fixes, patching security holes, and adding features. Because your developers need to get their work done, they make changes to their internal fork of the project as necessary—but they don't reap any of the benefits of the changes that are happening upstream. Nor, in fact, does upstream benefit from any of your development work. As shown in the figure that follows, these are now two parallel lines that won't meet unless someone puts a great deal of intentional effort into it.

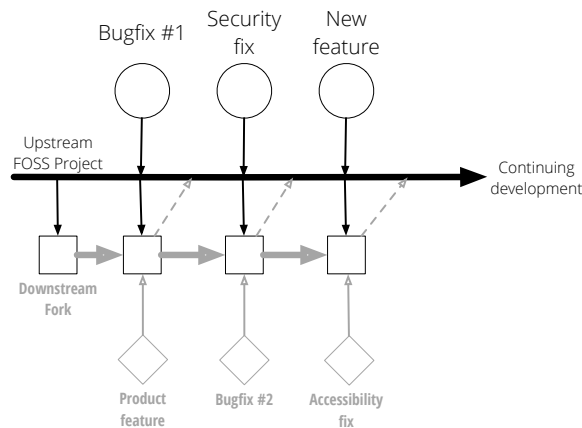


A company with an internal fork has some options for how to proceed. First, they could have their developers manually apply any upstream fixes and features to the internally maintained fork of the project. This can help to keep the fork up-to-date with upstream, but it's a lot of work and usually introduces complexity and risk.

Second—and perhaps concurrently—your developers can *forward port* the changes on the internal fork up to the upstream project. Again, this can be a lot of work. It also may not be welcomed by the upstream FOSS community. Any contribution it accepts, the community tacitly agrees it will support and maintain. Also, some of the features your developers add may not be in line

with the roadmap for the FOSS project. In both of these situations, it's possible the community will reject the contribution (*patch*).

The final option is also the accepted best practice: your company operates according to an *upstream-first policy*. In this policy, it's OK for your company to maintain an internal *clone* of the FOSS project, which it regularly syncs up with the upstream source, including contributing any improvements your developers make (refresh your cloning knowledge in [Do You Have the Spoons to Deal with a Fork?, on page 45](#)). This avoids the considerable time and expense required to maintain an internal fork while ensuring that everyone continues to benefit from the FOSS ecosystem. This is illustrated in the following figure:



It's still possible that your company may need to maintain an internal fork of a project for some reason. For instance, perhaps your product requires its user interface be available in Greek, but the project's internationalization process hasn't yet made it to that language. In this case, your company may need to maintain an internal fork for as long as it takes to translate the project and contribute that translation upstream. Whatever the reason, if your company does end up maintaining an internal fork of a FOSS project, it should be the exception and not the rule.

Intellectual Property Considerations

Before you start your outbound FOSS initiatives, make sure you understand the intellectual property (IP) risk profile of your organization. IP is a primary concern of many companies where outbound contributions are concerned. As you learned in [IP Exposure, on page 256](#), it also can be a sticking point that can make or break an outbound FOSS contribution effort.

Copyright

Copyright is likely to be the first IP topic you deal with when planning your outbound initiatives. Many of the contributions to FOSS projects may include IP that's copyrightable. Who owns that copyright in the case of contributions coming from your employees? If you remember from [Contributions, but on Whose Behalf?, on page 30](#), the answer may not be a simple one. Make sure you know that answer and prepare materials to support it before you launch your outbound initiatives.

For instance, many well-intentioned employees aren't aware of (or don't recall) your company's IP policies and may contribute to FOSS projects without permission, unintentionally sharing contributions for which they have no copyright. Some of these contributions could include patentable innovations, trade secrets, or in a worst-case scenario even login credentials for critical systems.

Thankfully, most contributions are fairly innocuous, like bug fixes that have no direct impact on product strategy, but that doesn't mean you can ignore the potential for leaking IP. It's best to assume most people will need refreshers on the organization's IP policies and prepare accordingly. Create documentation and training materials in advance and include them in the outbound initiative onboarding process that you'll learn about in [Onboard Outbound Contributors, on page 287](#). If you're clear about the policies and procedures up front, people will be less likely to make those well-intentioned mistakes.

CLA/DCO

Part of planning for copyright is knowing how your company handles Contributor Licensing Agreements (CLA, [Contributor License Agreements, on page 34](#)) and the Developer Certificate of Origin (DCO, [Developer Certificate of Origin, on page 35](#)). Each of these is a legal document that someone signs to declare that they have the right to share the things they're contributing to a FOSS project, but most contributors see these only as a hurdle slowing down their contributions. They agree to the CLA or DCO without reading or understanding it so they can get on with what they were doing.

Don't fault them for this behavior; we all do it. When faced with a pop-up box full of complex Terms of Service (ToS) or End User Licensing Agreement (EULA) text, do you *actually* read it or do you click the "I read this" checkbox and the "I agree" button as quickly as you can? If you're like most people, you click the thing and get on with your life. These walls of text are an *antipattern*, leading people into agreements they don't understand or aren't aware of.

So you may find that staff members will sign CLAs and DCOs without understanding that they're not actually authorized to do so on behalf of the company. How to handle CLAs and DCOs is something that should be covered in the outbound P&P and onboarding to minimize this mistake.

CCLA

One way to handle CLAs is to arrange to sign a Corporate Contributor Licensing Agreement (CCLA) with any FOSS projects that you're going to include in your outbound contribution initiatives. This is a single CLA agreement that gives blanket permission to anyone from your organization to contribute to the project. Note that if a project doesn't already use a CLA at all, it's not likely to start using a CCLA purely for the convenience of your organization. A lot of administrative overhead goes into maintaining a CLA process, and some projects don't think it's worth the trouble. Those that already use a CLA, though, may gladly provide a single CCLA that will cover many different contributors. If your company has signed CCLAs with any projects, make sure you include a list of those in your P&P and in your outbound onboarding process.

Patents

Patents, if you recall from [Patents, on page 21](#), protect inventions. Specifically, the invention must be unique and nonobvious, which is to say no one must have created it before (*prior art*) and someone with an "ordinary" level of skill in the industry wouldn't think that the invention is obvious (not innovative).

It's possible that some of the outbound contributions your organization makes to FOSS projects may be patentable. For some companies this is no big deal. Other companies, however, care a lot. These companies rely on protecting and licensing their patent portfolio as a key element of their business models.

Companies that place so much importance on patents may wish to add another layer of due diligence prior to making any outbound FOSS contributions. This may include requiring that all potential contributions receive a patent review prior to submitting them to the FOSS project. It also may include paying closer attention to the license on the FOSS project. Does that license have a patent clause (which you learned about in [Patent Clauses, on page 22](#))? This detail may be important for your company, so make sure you check with the patent specialists on your legal team prior to making contributions.

If it turns out that outbound contributions need patent review prior to submission, people may see it as a burden. This may result in fewer contributions. To reduce that perception, make sure everyone knows the reason for the review and they understand it's not simply busy work for the sake of busy

work. Also, see whether it's possible to create a simple threshold or heuristic to reduce the number of reviews. You could institute the following, for instance:

- If the contribution is not source code, then it can skip review; only standard processes apply.
- If it's source code but is a fix to an existing feature, then it can skip review; only standard processes apply.
- If it's source code and is a new feature, then it must receive IP review; standard processes apply after the contribution passes review.

Alternatively, if your organization knows it'll be contributing patentable work to a FOSS project, it may wish to consider either joining or creating a patent pool for it or otherwise making a public pledge not to enforce patents on that work if used in context of the project (see [Everyone into the Patent Pool, on page 23](#)). Work with your legal team to determine what sort of process is right for your company.

FOSS Patent Sensitivity

While software is eligible for patents in many jurisdictions, many in the software industry and especially in the FOSS ecosystem strongly believe that software *should not* be patented at all in any jurisdiction.

This belief can add tension or conflict when a FOSS community discovers a company that's contributing to the project is also taking out patents over their contributed code. It can be OK to do this (and can be a strong business strategy) but must be approached with care and empathy when a FOSS community is involved.

Prior to making contributions of patented software, try to determine how the community may react. If they're sensitive to the concept of patented software in the FOSS project, then take steps to communicate with them early on, letting them know what your company will be doing and why.

Remember: this project is likely important to your organization's business operations. It would be unwise to offend those who maintain it. Approach the matter with empathy; their concerns are just as valid as your company's.

Trademarks

The name of your organization—and perhaps even of your products—is undoubtedly a registered trademark in at least one jurisdiction. The same can be said for most organizations and also for FOSS projects. In fact, many FOSS projects have policies in place declaring their trademarks and how they can be used. For instance, the trademark policy of the Python

Software Foundation—warden of the Python programming language and its ecosystem—clearly states approved uses for marks it has registered:¹⁰

...stating accurately that software is written in the Python programming language, that it is compatible with the Python programming language, or that it contains the Python programming language, is always allowed. In those cases, you may use the word “Python” or the unaltered logos to indicate this, without our prior approval. This is true both for non-commercial and commercial uses.

This policy includes several other clauses, including defining uses that are never allowed and those that require approval as well as royalties. It’s a good example of a thorough FOSS trademark policy.

As you plan your outbound FOSS initiatives, be aware of not only the trademarks your company holds but also those of other companies and of the FOSS projects themselves. Ensure that trademark handling is a part of any outbound FOSS onboarding and related P&P. The safest form of trademark handling is not to use or mention trademarks at all in outbound FOSS contributions, but in some cases it may make sense. For instance, you may wish to give credit to a company or project for inspiring an idea that your organization then adds to an open source project. Consult with your legal team to develop clear and simple guidelines.

Verify Your Corporate FOSS Contribution Policies

In [Chapter 15, Outbound Policies and Procedures, on page 233](#), you learned about and perhaps even created policies and procedures (P&P) for your organization’s outbound FOSS engagements. If you haven’t created them yet, now’s the time to get that done. If you’ve already made a start on them, good for you! Regardless of where you are in the P&P process, double-check the finished product before you actually launch any initiatives.

- Do the policies tell people not only what to do but why?
- Do you have procedures for all the steps in the outbound contribution process?
- Does each document include a way for people to contact someone if they have questions or problems?

Don’t only review the documents yourself or only have your team do it. Get people with different perspectives to give the documents a test read. They’ll undoubtedly find some areas that need clarifying or are missing altogether.

10. <https://www.python.org/psf/trademarks/>

Have a member of your friendly, neighborhood legal team give it a look. Taking a few days to do this verification before you launch any initiatives can save a lot of headaches and ensure your organization's outbound engagements move much more smoothly.

Contribution Oversight

Depending upon the business needs of your company, part of the outbound FOSS contribution process may include oversight, review, or sign-off by others prior to making a contribution public. For instance, in [Patents, on page 275](#), you learned about the business need of some companies to do an IP review of contributions to FOSS projects. It's also common for companies to require other sorts of review, including these:

- Security and authorization
- Quality of contribution
- License and copyright statement

The trick with any sort of review process is to make sure it doesn't slow down the contribution too much, otherwise it'll discourage participation in outbound initiatives. The best way to do this is to make sure that anyone who is performing reviews is guaranteed the time required to do so and that how well they perform their review duties is a part of their performance assessment.

I can't stress enough that these reviews are not in addition to the reviewers' existing duties but may in fact replace some of those duties. People whose managers overload them with work will, through no fault of their own, do a poor job. If these reviews are important enough to your company's business that they're required for contributions, then they're important enough to staff properly. Otherwise the review policy is a performative sham.

Track Company Contributions

Keeping track of where and how your company is engaging with FOSS projects is a concern that too many companies leave until after they've already started contributing. In fact, it's something that you should plan for from the very start of your outbound FOSS initiatives. This is valuable information and can be more difficult to start collecting after the fact.

The following are some reasons why your organization might want to keep track of its outbound engagements:

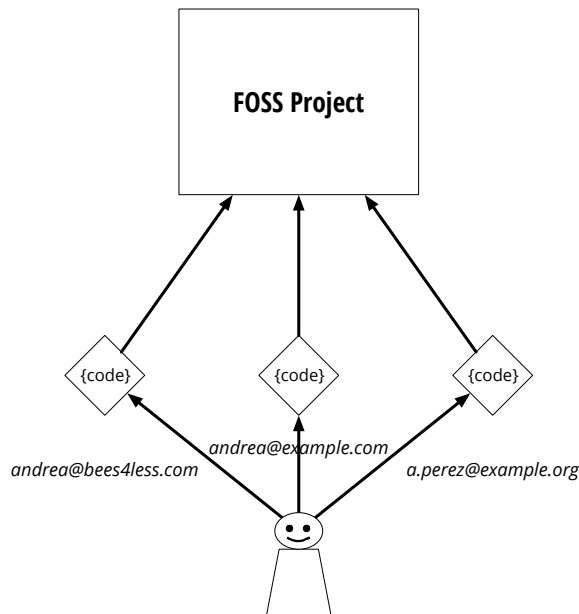
- Verifying that the organization is supporting critical FOSS projects and that this support is effective

- Estimating impact the contributions are having on business operations
- Evaluating contributions for possible tax implications
- Ensuring company is getting appropriate credit and copyright for its contributions
- Using the data for recruiting, marketing, budgeting, and other operational concerns

As you can see, many good reasons exist for why you should track where and how your company engages with upstream FOSS projects. The problem is that doing so is harder than it seems.

Easier Said Than Done

The problem lies in *identity management*. Let's say Bees4Less has an employment agreement that states that the company has copyright over any work their employees perform on company equipment. Andrea, a programmer at Bees4Less, is working on fixing a bug in a FOSS project that's critical to Bees4Less business operations. Because she's a remote worker, she spends most of her time on her Bees4Less laptop. It's too much of a hassle to dig out her own, and besides, the Bees4Less laptop is faster and more powerful. The bug fix requires she make three contributions to the project, but because she's making them at different times, she ends up using different email addresses (read: *identities*) to make each contribution (see the following figure).



On the receiving end it's not easy to associate all of those contributions to one person. That's because the Git version control tooling used by most FOSS projects uses email address as the primary means of identifying contributors. From the perspective of Bees4Less, these are all contributions from a single person on behalf of the company. But from the perspective of the project, these are three contributions from three different identities (email addresses).

Andrea herself can work around this in various ways (of course, this will require she knows the work-around is necessary in the first place). The Git tooling allows people to configure which email address to use for contributions, and that email address can even vary by project.¹¹ Most of the popular online code forges, such as GitHub¹² and GitLab,¹³ also support configuring and selecting different email addresses for different purposes. Using these tools, Andrea can set up her Bees4Less laptop so it always uses her `andrea@bees4less.com` email address for any contributions she makes from that computer. This can make it significantly easier for Bees4Less to see who is contributing to projects on their behalf.

A Common Misunderstanding

This may seem like a contrived example, but in fact it's something that happens all the time in practice. Developers frequently contribute to a FOSS project using different email addresses and just as frequently use company equipment for personal projects. Often, this isn't a problem.

However, in this case, because of the IP policy in the Bees4Less employment agreement, this practice may cause some headaches. For instance, Andrea undoubtedly is unaware of or has forgotten this policy. She may feel that of the three contributions she made, only one of them was actually on behalf of the company. She also may be creating or contributing to other FOSS projects using the Bees4Less laptop in her free time. Because of the policy, the copyright for all of that work may in fact belong to Bees4Less. As you can see, things can get complicated quickly and, depending on how Bees4Less decides to handle the situation, could involve legal action. It could get ugly.

This is why it's important not only to understand the IP policies to which the staff have agreed but also to remind *them* of the policies. If you can also work to make the policies as clear and as unrestrictive as possible, that will help avoid a lot of confusion and missteps. Revisit [Open the Doors to Open Source Contributions, on page 237](#), to learn more about creating open employment agreements.

11. <https://git-scm.com/docs/git-config>

12. <https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address>

13. <https://docs.gitlab.com/ee/user/profile/>

Corporate Contribution Metrics

As you determine what data to collect and track about your outbound FOSS initiatives, don't forget what you learned in [Be Clear About Your Goals, on page 243](#): always keep the goals of the initiatives front and center. If the metrics you track don't align with what you're trying to accomplish with the initiatives, you're not only tracking the wrong things but also wasting a lot of time and effort.

Here are some example goals and a selection of associated metrics:

Influence direction of the project	• Number of contributors from your organization • Number of core contributors from your organization
Support project sustainability	• Number of overall contributors to the project • Number of organizations contributing to the project • Mean time from ticket open to closing • Mean time for contribution review and merge
Hire more qualified people more quickly	• Number of job applicants drawn from project community • Mean time from job posting to hire for project-relevant roles • Mean time from onboarding to productivity for project-relevant roles

These are only a few of the possible goals and metrics; the actual ones selected will depend entirely upon the business needs of your company. This means that the possibilities for goals and metrics are endless, which can seem a bit daunting at first.

Thankfully, there's help. The Community Health Analytics in Open Source Software (CHAOSS) project has defined and collected a knowledge base of dozens of the most common and relevant metrics for FOSS engagements of all sorts.¹⁴ Browsing this collection can help to inspire and focus your metric-defining efforts.

14. <https://chaoss.community/kb-metrics-and-metrics-models/>

Aim for Good Enough Tracking

In [Easier Said Than Done](#), on page 279, you learned how difficult it can be to collect some of the data you may need for the metrics you wish to track. Don't let that stop you from trying. You may never be able to collect complete data, and that's OK. Perfect is the enemy of good enough, which is all you really need. Good-enough data can still show you trends to get a sense of whether you're on course toward the goals you've set for your outbound FOSS initiatives.

It can be tempting to shout, à la Allie Brosh,¹⁵ “COLLECT ALL THE DATA!” and track every possible metric right from the start. Do your best to resist this temptation. You have enough to worry about with launching a whole new initiative. Don't add data wrangling to your troubles.

From among all of the possible combinations of goals and metrics for your organization, try to select the small handful that are the most meaningful for your initiatives and will resonate well with company leadership. What are the metrics they care about the most? Start by collecting data for and tracking only those metrics.

The smaller scope of your metric-tracking efforts will allow people to focus and produce better results as they get up to speed with outbound FOSS. Later, after the initiatives have had time to settle in and the team has had the chance to gauge what's most important, you can add additional metrics to those you're already tracking.

Collecting, storing, and presenting all of this data can be a royal pain, but thankfully you don't need to reinvent the metrics-collecting wheel. Aside from providing a knowledge base of metrics, the CHAOSS community also develops software for collecting and visualizing data from various community sources, such as version control systems, mailing lists, and chat rooms.¹⁶

Your organization can run this software locally on your own servers, or you can contract with a company like Bitergia to both run the software and provide professional advice.¹⁷

Now that you've learned the fundamentals that will help your outbound FOSS efforts to be successful, it's time to develop your contribution strategy.

15. https://en.wikipedia.org/wiki/Allie_Brosh

16. <https://chaoss.community/software/>

17. <https://bitergia.com>

Develop Your Contribution Strategies

Don't follow the pack. Take the time to figure out which types of contributions are right for your strategy *and* for the projects most important to you. This isn't the matter of a moment, especially not if you'd like your outbound FOSS strategy to be successful. Like many things worth doing in life, this one breaks down into a few distinct steps:

1. Identify constraints.
2. Identify critical projects.
3. Identify the needs of those projects.
4. Define your contribution strategy.
5. Review P&P.
6. Prepare onboarding.
7. Get the word out.
8. Start contributing!

Now, let's see how to apply those steps when choosing what contributions are right for your business.

Identify Your Internal Constraints

Before diving into any outbound FOSS contribution program, always look internally first. What constraints will you be working with inside your organization? If you don't learn about these before you get started, you're setting the initiative up for failure, and failed initiatives don't often get a second chance.

The Budget Constraint

The constraint people usually think of first is budget: how much money do we think we'll need, and how much is available?

If you've ever worked with budgets, you know that a lot of complexity hides behind those two deceptively simple questions. Their answers also are inter-dependent. You may be able to use your Need estimate to influence how much your company makes Available. Conversely, their Available answer certainly will affect the scope of your Need answer.

Resolving this budget question is a process that will be ongoing, but you'll need to start that resolution from the very beginning. Like all of the internal constraints, this one will directly impact the entire initiative going forward.

The Time Constraint

Another common constraint is time. I'm including staff availability in this constraint, since that's really what it means: who is available (has the time) to do this work? Will you need to reassign staff from other teams or projects? Will you need to plan how to convince other managers to allow their staff the time to contribute? If you don't consider this constraint until the end, all your work may come to a sudden and unpleasant stop.

The Skills Constraint

Related to time is the constraint of skills. It doesn't matter how many people are available to help with the initiative if they don't know how to do the work that the program requires.

Will you need to include time to train people, or time to hire people who already have the skills, and then more time to onboard them into the company? If your company doesn't yet have an OSPO, you may need to start one, and that doesn't happen overnight.

Unfortunately, the constraint of skills regularly gets brushed off as a nice-to-have. Too often I've seen managers throw people into FOSS (and other) initiatives with a "they'll figure it out" attitude. This rarely works and usually ends with wasted time and budget as well as high employee turnover.

The Strategy Constraint

From an employment perspective, lack of alignment with business strategies is an existential crisis. It's also bad business to fund initiatives that provide no value to the organization.

If the business strategy of your company or organization isn't headed in the same direction as the outbound FOSS initiatives, it'll be practically impossible to launch and maintain a successful contribution effort. Everyone involved with the initiative must be aware of how their efforts support the strategy and

be ready at any moment to answer the question, “How does this help the business?” Revisit [Initiative Value: If You Know It, Show It, on page 244](#), for guidance on this.

Even if the initiative supports the strategies and priorities of today, the entire picture could change next year, next quarter, or even tomorrow. Always work to stay aware of strategic priorities for the organization to ensure ongoing initiatives continue to provide the necessary support.

The Leadership Support Constraint

A constraint that people regularly take for granted is leadership support. It may be obvious to *you* that outbound contributions are the right thing to do, but if you don’t convince other leaders that the plan provides value to the business, then that plan is unlikely to get very far. Yes, this holds true even if you’re sitting in the C-suite. As you’ve already seen, FOSS is collaborative, and no software succeeds through application of a single role. As well, studies show that all initiatives benefit from broad support across the organization.¹

Identify Critical Projects

In [The Process \(Roughly\), on page 182](#), you learned how to do software composition analysis (SCA) on your free and open source software supply chain (SSC). As a part of this, you learned to [Reduce Scope by Focusing on Load-Bearing Links, on page 188](#). These load-bearing links are your critical FOSS projects.

It’s possible your business will have more load-bearing links than you’re able to contribute to at first, and that’s OK. Nothing’s wrong with starting small and then scaling up your contribution process as you’re able to prove its success and value to the business.

However, you have no good way to tell how many projects your company is able to support with contributions unless you understand the needs of those critical projects. Once you know that, you’ll be able to see how many of those needs you can afford to meet. That’s your next step.

Identify Project Needs

One size doesn’t fit all when it comes to meeting the needs of critical projects. You can see this at work in your own teams: each one has its own unique pain points. What works to move one team forward won’t work for another team with different needs. The same holds true for FOSS projects.

1. <https://hbr.org/2018/11/making-process-improvements-stick>

Discover the needs of critical projects before you start your contribution initiative. Yes, this can add some time to the overall effort, but it also helps to guarantee that your company's contributions provide maximum value for all stakeholders. That relatively small investment in time up front can therefore pay large dividends.

It's not difficult to discover these project needs. Here are some approaches you can take:

- Review the project issue tracker for ideas or recurring themes.
- Join the mailing list and see what problems people discuss.
- Lurk on their real-time chat system and read the questions people ask.

These are effective ways to gauge a project's pain points, and they're not exclusive. You can combine them and include other approaches as make sense for the critical projects on your list.

Just Talk to Them

The best and most efficient way to discover the needs of a project on your critical list, however, is to *talk to them*. It's as simple as that; have your OSPO reach out to project maintainers and ask where your company's contributions would make the most difference for them.

It's surprising how few companies consider this last discovery option and instead send unnecessary contributions to projects. This approach has (at least) three problems. First of all, it's a waste of talent, treasure, and especially of time. Secondly, it's also arrogant of an outside entity to assume it knows a project's needs better than the project itself does. Finally, it's rude to force unrequested assistance on those who don't need it.

FOSS communities are spaces for collaboration, not dictators. Enter into your outbound FOSS engagement initiative with this collaborative mindset at the start, and you're much more likely to find that it—and the critical FOSS projects on which your company relies—will be successful.

Define Your Contribution Strategy

You know your constraints, your critical projects, and their needs. Now it's time to use all this information to decide where and how your company can contribute.

Don't forget that these contributions can take many different forms. You could combine time with treasure, time with talent, or time, talent, and treasure

together. It all depends upon the constraints within which you're working and the project needs you're aiming to meet.

The only wrong way to contribute is not to do it at all. This is the business equivalent of hiding under a blanket and hoping everything will work out. As powerful as it is, hope is not a good business strategy. The only way to ensure the strength of your FOSS software supply chain—and therefore of your business—is to take action. That action doesn't need to be big and impressive; even small actions matter and add up. Baby steps are still steps, and get you closer to your destination than if you were standing still.

Ensure Policies and Procedures Are in Place

At this point in your planning you're probably eager to get the contributing party started, but I encourage you to pause for a moment. Now that you know *what* you're going to do, you're ready to make sure people know *how* to do it.

Take some time to review policies and procedures (P&P) to verify that they'll support your plan. Can people read the P&P to learn the process of contributing and answer the most common questions? If the answer isn't there, is there a way for them to get their questions answered?

If you invest some effort in verifying P&P up front, before you launch your outbound contribution initiatives, you'll likely discover and remove possible roadblocks even before you start your journey. You and the staff will face less frustration, meet with more results along the path, and your initiatives are more likely to be successful.

Onboard Outbound Contributors

If working with and contributing to free and open source software were straightforward, I wouldn't be writing my second book on the subject. Asking people to work in an unfamiliar area and then expecting them to be successful all on their own is foolhardy at best and business malpractice at worst. If you want your initiative to be successful, you have to provide the training and support to set participants up for success. We provide onboarding materials for people who are new to the company, and there's no reason why we can't do the same for people who are new to an initiative.

This onboarding will take the form that makes the most sense for your organization. For instance, if there's a training team that assembles and publishes online training modules for internal use, work with them to develop a FOSS contributing module and make it available to all potential participants. Follow

the established documentation and training paths in your org; don't come up with something new that no one is able to remember or locate.

Regardless of the form of the onboarding, you should always provide a static resource to which people can return if they have questions. This resource goes by many names, but I call it the Getting Started Guide.

Prepare a Getting Started Guide

We've all come across Getting Started Guides (GSG), so the concept won't be a new one to you. You've likely found these are helpful to have around. Despite that, few companies invest the time needed to create them—or a lot of other helpful documentation. That's a shame since—again, like most documentation—it pays for itself almost immediately by allowing people to answer their own questions and be productive rapidly.

The outbound contribution GSG should include anything that people might ask about when participating in FOSS contributions. This can include the following:

- Which projects people may contribute to and why the company chose them
- How to set up their version control system so it uses the correct email address for contributions and why this matters
- A walk-through of the contribution process along with a way for them to find easy contributions when they're getting started

The lessons from [Make Policies Useful \(and Used\), on page 147](#), apply to the GSG as well. This includes having a way for participants to ask questions in case anything in the guide isn't clear or is entirely missing. In [The Edge Cases, on page 238](#), you learned that a ticketing system can be helpful for P&P questions, and the same holds true for the contributing GSG. Whatever the implementation, always make sure you don't leave people hanging when they have questions. Doing so will create ill will, and people will be less likely to participate in the initiative.

Get the Word Out

Communication will be key as you launch your initiatives. You want to not only reach out to the people who'll directly work with or on the initiatives but also to promote these efforts across the entire organization. Remember [Outbound Impacts the Entire Business Organization, on page 229](#), and [Outbound Isn't Limited to Code, on page 231](#). The wider you're able to spread

word of the initiatives, the more inclusive you can be of skill sets across the organization and the more the company as a whole benefits. Also, politically speaking, it's never a bad thing to be working on an initiative that has leadership support and is visible across the entire organization.

You'll find many ways to let others know about the initiatives. Discover the methods teams across the company use to share information, and work with them to mention the outbound FOSS initiatives. If your company has an internal communications team, they'll be invaluable for their knowledge and ability to share news at a higher level across the company. Arrange to join the regular staff meetings of relevant teams and make a short pitch or presentation. Get creative, and soon you'll have a well-promoted initiative and plenty of people interested in helping and seeing it succeed.

Start Contributing!

Now you're prepared to start your outbound FOSS contribution initiatives! Congratulations! Your outbound initiatives are now considerably more likely to succeed than most I've seen throughout my long career. Here's what you've accomplished so far:

1. You've identified your constraints and come up with a plan that works within them.
2. You've identified the projects to which your organization will contribute and what their needs are.
3. You've verified P&P and provided resources that will help—not block—people as they contribute.

Let's see how this may all look in practice.

The Bees4Less OSPO Contribution Strategy

This has been a lot to digest in only a few pages, so let's go through an example to help you better understand the process. To further help, in [Appendix 4, Worksheets and Resources, on page 419](#), you can find a worksheet for [Outbound FOSS Contribution Planning, on page 421](#).

We'll return to our friends at Bees4Less. Like so many of us, they're going through a digital transformation effort. A part of that is a growing awareness of and concern for the sustainability of the FOSS links in their software supply chain. Their leadership has tasked the Bees4Less OSPO to develop and launch an outbound FOSS contribution effort to make sure some business-critical projects remain stable.

ID Constraints

Bees4Less got lucky in their constraints: the decision to start supporting critical projects came directly from the VP of Product, who also says she'll be the executive sponsor for the entire initiative and give it some starting funding of USD 25,000 out of her own budget. That's great news, as it removes a few constraints for the team: they have leadership support, a budget, and know that their initiative is in line with current business strategy. Moreover, it's a priority project for the VP of Product, which shields it from a lot of complications with office politics.

That leaves the constraints of time and skills. The VP would like to see a report by Q4 of the 2024 fiscal year. Since Bees4Less has a fiscal year that starts in April, that means the OSPO has from February 2024 through January 2025 to launch their initiative and start showing results. While that's nearly a year, it's still not a lot of time, so the team decides to keep their first efforts small.

However, they don't know what the projects are yet, which is making it difficult for them to determine their skills constraints. They decide to return to this question after they choose the two projects where they'll get started.

Identify Critical Projects

As the OSPO team reviewed the FOSS dependencies used by Bees4Less, the VP of Product came to them with new information. The company is going to create a new product providing an analytics portal for commercial beekeepers to track the health and productivity of their hives. The product will be based on the Drupal Free Software project and use the Python pandas library in generating the analytics.

Since the company is going to need a lot of people with Drupal and pandas experience to maintain and grow this product offering, and because both of the communities are known to be friendly, be welcoming, and have a lot of good documentation for new contributors, the OSPO decided that these two critical projects would be a good place to start the contribution initiatives. There's no shame in starting something new on easy mode, after all.

This allowed them to review that skills constraint. Since everyone in Bees4Less who knows how to program in PHP will be needed to work on the new product, the company wouldn't easily be able to contribute code to Drupal and would have to find other ways to contribute there.

Identify the Needs of Those Projects

Both Drupal and pandas have excellent documentation that guides new contributors in how the projects function and how everyone can make a difference. This made the task of identifying project needs trivial for the Bees4Less OSPO team. They all marveled at how they seemed to have all green lights on their road toward outbound FOSS engagement.

Define Your Contribution Strategy

Looking at their constraints and the project needs, the OSPO took different contribution strategies for each of the projects.

Drupal

Since the company couldn't contribute software patches or features for Drupal, the team decided to use some of its budget to sponsor diversity, equity, and inclusion (DEI) tickets for DrupalCon, the big community Drupal conference. They would also work with the Drupal Association to pay for the travel and hotel for these DEI ticket holders to attend the event. In return, Bees4Less could have a booth spot in the expo hall at the conference. The OSPO was able to speak with the Bees4Less senior director of Documentation and secure some dedicated time from Aditya, a technical writer who'd expressed interest in learning more about open source. Aditya would be able to spend half his time contributing documentation and editing skills to the Drupal project.

pandas

For pandas, the OSPO team had no trouble at all finding willing programming volunteers to spend half of their time contributing software patches to the project. The directors of Product Development assured the OSPO that they'd figure it out and promised that they'd have a person who could dedicate half of their time to pandas. That person may change every quarter (so more people could get the experience), but the donated time would not. The OSPO team also chose to spend some of their budget on sponsoring Outreachy interns for the pandas project.²

Both

For both projects, the team decided they'd work with the projects to find internal volunteers to help with translating documentation. They'd also use their remaining budget to join the nonprofits that support these two projects,

2. <http://outreachy.org>

the Drupal Association and NumFOCUS. Since both of these organizations and Outreachy are 501(c)(3) nonprofits,³ they made a note to coordinate with the folks over in Finance in case there are positive tax implications from any of their monetary contributions.

Review P&P

Since the OSPO maintained all FOSS-related P&P at Bees4Less, they already knew the state of their policies and procedures, thanks to tracking all requests in a ticketing system. Everything was mostly OK, but they found some good contribution tips in the projects' documentation, and they wanted to include direct links to those in the internal documentation. They wrote up a ticket to do this work so they wouldn't lose track of the task.

Thankfully, neither of the projects require signing a CLA or CCLA prior to making a contribution, so the OSPO team was able to defer that sticky question until later. Still, they put a meeting on the calendar with their friends in Legal so they can start thinking about this sooner rather than later. Everything was going so smoothly! Unbelievable!

Prepare Onboarding

It would be relatively quick for the team to pull together the information they'd need for a Getting Started Guide (GSG) and initiative onboarding, but they'd still need to put those resources into a coherent format. Sky and Bear, two of the program managers in the OSPO, volunteered to assist with this, including coordinating with the team over in Career Development to create and publish some new training modules.

Get the Word Out

The Spring 2024 Bees4Less All-Hands meeting was coming up, and the OSPO didn't want to miss the opportunity to create some buzz. The team wasn't going to be ready to launch the initiative yet, but they arranged for the VP of Product to announce the upcoming initiative along with the 2024 product plans. They also asked the internal comms team whether they could place an article about the initiative in the internal newsletter that would be sent out soon after the all-hands. The comms team knew good, free content when they saw it and not only agreed to that article but also offered a regular newsletter column for the OSPO to share what's going on. This was too good an opportunity to pass up, so naturally the OSPO team gleefully accepted it.

3. [https://en.wikipedia.org/wiki/501\(c\)_organization#501\(c\)\(3\)](https://en.wikipedia.org/wiki/501(c)_organization#501(c)(3))

Get Contributing!

All of the planning is done! Here's what the Bees4Less team came up with when they filled in the worksheet:



Budget

\$25K USD, 1 FTE total from ProdDev teams



Time

Report first results for initiative by Q4 2024



Skills

Can't get time of any PHP devs



Strategy

Launching new customer portal based on Drupal in Q1 2025



Leadership

VP Product says will be exec sponsor

Outbound FOSS Planning

Date: Feb 29 2024

Drupal

pandas

CRITICAL PROJECTS

Funding

Documentation

Translations

Bug fixes

Translations

PROJECT NEEDS

Sponsor DEI tix @ DrupalCon

Assign .5 FTE writer/editor

Ask for translation volunteers

Assign .5 FTE Python Dev

Sponsor Outreachy intern

Ask for translation volunteers

CONTRIBUTION STRATEGY

Irene in Legal to review CLA policy with Jo in OSPO

Bear coordinating w/Career Dev to get training posted

Sky drafting onboarding docs and training modules

P & P AND ONBOARDING

VP Product to announce at Spring all-hands

OSPO now has a monthly column in company newsletter

OSPO w/follow up in company newsletter

Chat room for questions and community building

COMMUNICATION

Naturally this is only the first iteration of the plan. In the real world, the process would not go this smoothly, and the end product would go through several iterations in its lifetime as mistakes and assumptions are identified. No one and nothing is perfect, after all, but with a framework in place to ensure you've considered all the important subjects, the plan is much more likely to succeed.

All Big Ideas Start Small

The Bees4Less OSPO looked at their constraints and recognized that they wouldn't allow for an initiative with too large a scope.

It can be tempting to dive in and make a big, impressive plan, but I encourage you to resist that temptation. If your company is just getting started with its outbound FOSS contributions, you may want to consider starting small. Run a pilot program, a time- and scope-limited contribution effort to a single FOSS project. Learn from this program, iterate on your approach using this knowledge, and then scale up to a larger effort.

Rely on Your OSPO

The Open Source Program Office (OSPO) is an irreplaceable cog in this entire process, from initial planning straight through to execution and follow-ups. It's no mistake that the [Functions of an OSPO, on page 103](#), are exactly what's needed to support outbound FOSS initiatives as well as inbound.

If your company has an OSPO, it will be the group planning and driving your organization's outbound FOSS engagement, coordinating with other functions and stakeholders. As you can see, a lot of specialized work is involved with a successful outbound contribution initiative, and considering the importance of FOSS to the operations of most companies, it makes sense to have a team dedicated to this work.

However, it's still possible to have a successful outbound FOSS contribution initiative without an OSPO. It's harder, yes, but possible. Usually, making the effort successful in these conditions requires some staff be partially or entirely reassigned from their other duties to free them up to dedicate the necessary time to the initiative. It also requires these staff have the specialized experience and skills for the task.

If your company can't find qualified staff to reassign, or if it can't afford to reassign those staff away from their existing duties, it may want to consider hiring for an OSPO or contracting with outside consultants to lay the groundwork for the initiative and train staff to continue the work when their contract ends.

Remember, you're launching this initiative because it's strategic for the company and helps move it toward its goals. If the company can't or won't staff the initiative so it'll have any hope of success, it's worth asking whether those goals are realistic.

So far, the outbound engagements we've covered have centered around contributing to FOSS projects and communities. While that's an important form of outbound FOSS engagement, it's not the only one: your company can also release its own FOSS projects. Why on earth would it create something and then give it away for free? It turns out there are a lot of strategic advantages to doing so. You'll learn about the strategy of releasing FOSS projects in the next chapter.

The Strategy of Releasing FOSS

Contributing to FOSS communities and projects can be great for your business, but it's only one side of the outbound FOSS coin. The other side is releasing projects under free and open licenses.

Like making contributions, releasing projects can be a powerful tool in your strategy tool chest. You'll learn all about it in these next two chapters.

What It Means to Release a Project

When your company releases a FOSS project, it isn't enough to make it publicly available. It's also important that it be available under a license that's been approved as a free/open license by a standards body. You learned about these standards bodies in [What Makes a FOSS License “Open Source”, on page 50](#), and [Other Types of Licenses, on page 54](#), and in the next chapter you'll learn how to apply these licenses correctly to the projects your company releases.

It's now common to find organizations of all sizes releasing FOSS projects. Some examples include these:

- Netflix has released `bpftop`,¹ a tool for viewing the performance statistics of eBPF programs.
- Porsche released its design system for the web presence of Porsche brands and products.²
- The U.K. Government released the entirety of the `uk.gov` website for other governments to repurpose.³

1. <https://github.com/Netflix/bpftop>

2. <https://github.com/porsche-design-system/porsche-design-system>

3. <https://github.com/alphagov>

If you look at the URLs for these projects, you'll notice a similarity: each of them is hosted on the GitHub collaboration forge. Other forges are available, such as GitLab or Bitbucket, but at the moment GitHub is the most popular. Popular doesn't necessarily mean "best for your purposes" though. In the next chapter we'll talk about how to select the right forge for your organization's projects.

Releasing a project doesn't necessarily require that your organization continue maintaining it or even support it in any way. In [Throw It Over the Wall, on page 308](#), you'll learn about the style of releasing commonly known as "throwing it over the wall." While this is a valid way to release projects, in general people expect that a project has someone keeping an eye on it, answering questions, fixing bugs, and just generally enabling the project to continue being a going concern. As you start learning about releasing FOSS projects, keep this "care and feeding" in mind, as it'll require investment of resources.

Why Organizations Release Projects

Releasing a FOSS project isn't a simple undertaking (not if you want to do it correctly, anyway). First your company needs to create the thing to begin with, then prepare it for release, release it, and perform any necessary follow-up, support, or maintenance. The next chapter walks you through the many moving parts of releasing a FOSS project.

Why would anyone put in all of this work just to give something away for free? It sounds weird and counterintuitive from a business perspective. Despite that, companies and other organizations release FOSS projects every single day. They don't do it because it's trendy—though, admittedly, at the moment it is. No, they do it because releasing FOSS projects has proven business value. Here are a few of the benefits it can bring to your organization.

Attention

Oscar Wilde, in *The Picture of Dorian Gray*, quipped that "there is only one thing in the world worse than being talked about, and that is not being talked about."⁴ Combine that with P. T. Barnum's attributed saying that "there's no such thing as bad publicity," and you're left with a nearly insatiable need to find ways to keep people talking about your organization. Some do this with clickbait headlines and inflammatory social media posts. Others do this by releasing projects that can help others.

4. <https://www.gutenberg.org/cache/epub/174/pg174-images.html#chap01>

Each project and resource your organization releases is yet another opportunity to make announcements and turn heads, usually in a good way. As you learned in [Reputation and Brand, on page 90](#), giving back to the community by freely sharing useful materials engenders and strengthens trust in your company and your brand.

Collaboration

When your company creates something to solve a problem it's having, it's very likely others are suffering from the same problem. You could convert the solution into a product offering, but usually that's neither possible nor advisable. The solution may be too small to productize, or it may not align with your organization's strategic product direction. Usually, solutions like these remain in-house only and usually end up collecting dust rather than living up to their full potential.

Releasing solutions as FOSS projects allows them to be used by many others, extending their usefulness and value. Most of those people will have different perspectives and use cases, yet all of them are able to benefit from the solution in some way. Even better, FOSS licensing and culture encourages people to contribute their improvements back, improving the solution for everyone. These improvements include the addition of not only innovative features but also fixes and other enhancements, such as translations. Each new contribution enriches the possibilities available to everyone.

By working collaboratively in the open, all share the benefits that come from a diversity of perspectives and the burdens of maintenance rather than any one of them having to do it all by themselves.

Coopetition

When collaboration includes your competitors, you get to call it by the trendy buzzword *coopetition*. Buzzword it may be, but this strategic alliance between competing organizations provides real value to all parties. By cooperating on the creation of a standard solution for an industry-wide problem, companies are free to focus more of their energies on differentiating factors in their businesses.

Sometimes you'll find situations where the coopetition is intentional. Organizations form a limited partnership to build a solution prior to releasing it to the public, at which time they work to recruit additional collaborators.

More often though, competitors discover they're coincidental cooperators. It's only natural that entities with common problems will gravitate toward and contribute to the same freely available solutions. Often, those entities operate in the same industry and may be in direct competition with each other. Because of this, many FOSS-related nonprofits enforce strict antitrust and anticompetition policies to ensure a safe and productive collaboration environment.⁵

Strategic Move

The aforementioned reasons all have their strategic applications, so releasing a FOSS project can be a powerful weapon in a competitive marketplace. Consider the following examples:

1. *Create a new line of business.* Perhaps the solution developed in [Collaboration, on page 299](#), solves a problem in a compelling enough way to warrant productization. If that product is software, then it can be offered as *software as a service* (SaaS). Once the product has necessary momentum, the company can gain attention, goodwill, and collaborators by releasing it under a FOSS license like the Affero General Public License (AGPL).⁶
2. *Cut off a competitor.* By releasing that solution under a reciprocal license, the company can discourage or even block competitors' entrants to the market. Yes, the competitor could release their own version of the AGPL'd SaaS solution, but overcoming the momentum and first-mover advantage of the originating organization can be very difficult. Often, doing so successfully relies on the originating firm doing a poor job with their SaaS product, allowing competitors to provide a more compelling offering of the solution.
3. *Own an ecosystem.* That first-mover advantage can be huge. If the solution is in a new or otherwise underdeveloped corner of the industry, the organization that releases it under a FOSS license can build strong awareness, bringing with it a vocal user base and community of early adopters. By getting to that corner first, the openly available solution becomes a standard against which all potential market entrants are judged.

You'll learn more about the strategic power of releasing FOSS throughout this chapter.

5. https://www.eclipse.org/org/documents/Eclipse_Antitrust_Policy.pdf

6. <https://www.gnu.org/licenses/agpl-3.0.en.html>

Charging for Free

Read any FOSS license. Really dig around in there. You'll never find any clause or statement that prohibits you from charging for material released under that license. As a matter of fact, prohibiting commercial use of FOSS-licensed materials violates Section 6 of the Open Source Definition:^a

6. No Discrimination Against Fields of Endeavor: The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Remember from [Basics of Licensing, on page 39](#): Licenses say what people can and cannot do with a copyrighted work. Typically, for FOSS-licensed work, people may do whatever they want with it as long as they provide credit to the original authors. If it's a reciprocal software license, they also must provide the corresponding source code and anything required to build the software.

It is absolutely OK to sell things that have been released under FOSS licenses. Heck, you don't even need to be the one to release it. I could release an image under a Creative Commons Attribution (CC BY) license,^b and you would be allowed to take that image and make it available for sale in whatever form you want—as long as you obey the terms of the license.

So there's no legal reason why you can't charge money for Free Software. However, if you decide to build your business around a FOSS project, you need to keep in mind always to make sure your business model and plan take the FOSS nature of the project into account. You'll learn more about this in [Chapter 22, Incorporate FOSS into Business Models, on page 369](#).

a. <https://opensource.org/osd>

b. <https://creativecommons.org/licenses/by/4.0/>

Release Strategically

[How Releasing FOSS Can Benefit Your Strategy, on page 249](#), and [Why Organizations Release Projects, on page 298](#), have provided you with a host of reasons how releasing FOSS projects may support a business strategy. Doing this is no harder than doing any other business strategy planning and implementation, but doing it successfully does require two things:

1. Have well-defined strategic goals.
2. Remember that releasing FOSS is a tool you have in your collection.

Set Goals

Goals are a recurring theme in this book (I count 51 instances of the word so far), and with good reason. A lack of well-defined goals is one of the most

common mistakes I see in organizations with which I work, second only to a lack of communication or understanding of those goals.

I get it; strategy and goals are hard. Thankfully, a lot of resources are out there to help you with it. You'll even find goal frameworks to give you a scaffolding on which to hang your strategic thoughts, including but certainly not limited to these:

- SMART goals⁷
- Locke and Latham's Five Principles⁸
- Objectives and Key Results (OKRs)⁹

If your organization chooses to use a framework, it doesn't really matter which one it is. They all get you closer to strategy and goals than having none at all. Every goal-setting framework has its vocal proponents, but the primary selection criteria shouldn't be framework popularity but rather whether the framework makes sense within the structure and culture of your organization. Awkwardly duct-taping an ill-fitting framework onto your culture because some high-priced consultant told you to never ends well. Do the research, evaluate the options, choose the one that resonates best with how your people think, communicate, and operate, and you're guaranteed to have more effective strategy and business goals.

Evaluate Your Strategic Tool Chest

This book may be your first exposure to using FOSS strategically within your company, so you may not yet have integrated it into your strategic thinking. This is common when learning something new. It takes a while for it to feel "normal" for us. Similarly, it's common to be so excited by the new learning that the urge to try to wedge that new thing into everything you do can be difficult to resist. When you have a new hammer, everything starts to look like a nail.

Rather than focusing on the Newest, Shiniest Tool in your collection, try to keep more of an overview perspective. What are the tools you have at your disposal? What can you use to shift operations toward the direction of the goals?

It can be helpful to visualize what tools you have in your strategic tool chest. Get a stack of note cards and put one strategy tool on each one. If you're in Sales, you might have cards that include Pricing, New Market, New Channel, Live Demos, and other sales-related tools. When you're working on coming

7. https://en.wikipedia.org/wiki/SMART_criteria

8. https://med.stanford.edu/content/dam/sm/s-spire/documents/PD.locke-and-latham-retrospective_Paper.pdf

9. https://en.wikipedia.org/wiki/Objectives_and_key_results

up with a strategy, lay out all the cards and use them to inspire your thinking. This can help prevent you from overlooking a potentially useful tool or getting fixated on any particular one.

This collection will differ for each leader, as they're all overseeing different company resources and therefore have different tools that they can bring to bear. This is why we have teams: we all bring different tools to the job, so we can build something great together.

Don't Limit Yourself to Software

Remember [Other Types of Licenses, on page 54](#)? Many of these other licenses enable the free and open sharing of copyrighted works beyond software. Whatever your company wishes to release to help it move toward its business goals, there's a free and open license for it:

- Use Creative Commons licenses to share creative works including art, writing, and audio-visual creations.¹⁰
- Apply Open Data Commons licenses to share data sets.¹¹
- Share schematics and similar designs according to the Open Source Hardware guidelines.¹²

Decide Whether to Release a Project

Should you release an internally developed project as FOSS? The answer isn't always yes. If you can't point to direct business value for the effort, releasing FOSS may not be the best choice for your organization.

Harken back to [Maintenance Costs, on page 83](#): free and open source projects are “free as in puppy.” When your organization releases a project, it has acquired a puppy, with all of the associated benefits and costs. If your company isn't prepared to do the right thing to keep the puppy healthy, it probably shouldn't get one.

Much like the decision you tackled in [Chapter 18, Develop Your Contribution Strategies, on page 283](#), you can break this one down into smaller bits so it's easier to make. Start by answering these two questions:

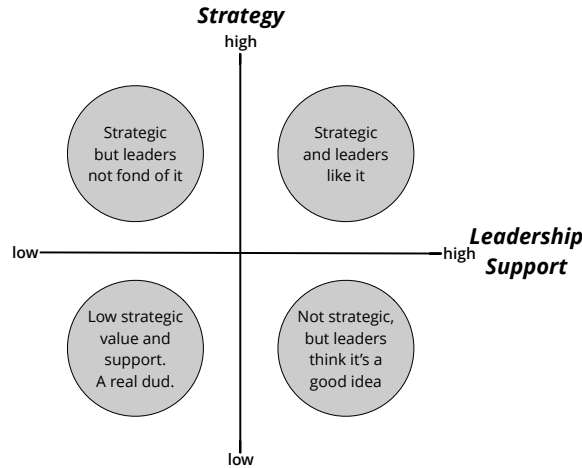
1. Does the release idea align with business strategy?
2. Do you have leadership support?

10. <https://creativecommons.org/share-your-work/cclicenses/>

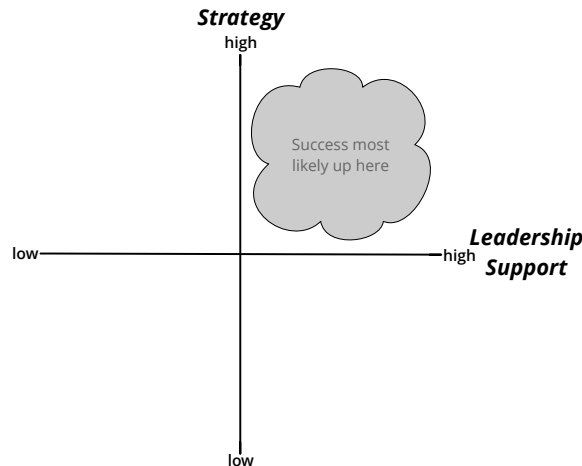
11. <https://opendatacommons.org>

12. <https://www.oshwa.org/sharing-best-practices/>

After doing the work in [Identify Your Internal Constraints, on page 283](#), you probably have a good idea of your constraints by now. Let's have a look at these two—strategy alignment and leadership support—by using the old business standby of a two-by-two diagram (2x2):



Putting these two constraints on the 2x2 and plotting where your idea is with respect to each of them can quickly reveal whether it makes sense to continue. Only one of the quadrants of the 2x2, the upper-right one, has the potential for both some sort of strategy alignment and even a moderate amount of leadership support (see the following figure). If the release idea, after plotting out where it stands with regard to these two constraints, doesn't land in that upper-right quadrant, I'd advise you to save yourself a lot of time and headaches and hold off on releasing the project.



This alone answers the question of whether you should continue chasing the idea of releasing a FOSS project, and you only needed to review two constraints to do it. If it's not aligned with business strategy and it has no support from leadership, it's unlikely to be successful.

What if you *do* end up in that upper-right quadrant though? Congratulations! You're on your way toward getting a new puppy! Now what breed of dog will you get?

FOSS and Export Regulations

When you release a software product, it has the potential to cross international borders. Therefore, depending upon your jurisdiction, government export regulations may apply to that software. In the United States, at least, you'll be glad to hear that export regulations *do not* apply to free and open source software projects. Mostly.

According to the Linux Foundation (LF) Legal Team,^a U.S. export regulations don't apply to any FOSS projects unless it includes nonstandard cryptography—which is to say, pretty much all FOSS projects are immune to export regulations.

Regardless of what you read here or in any publication from a nongovernmental organization, *always* work with your own legal team to verify the regulations in your local jurisdiction, even the United States. Due to various socio-economic-political factors, these regulations tend to change frequently. It's best not to assume you know what they say today, even if you had that information last year.

a. <https://www.linuxfoundation.org/resources/publications/understanding-us-export-controls-with-open-source-projects>

Choose What to Release

You should choose a puppy that suits the constraints of your lifestyle. Some dogs, like Akitas and Border Collies, need a lot of activity and attention to stay happy and healthy. Others, like a Bernese Mountain Dog or a Malamute, need a lot of space, if only for napping. You're smart, so you can see where I'm going with this: the project you select to release should similarly match the constraints with which you're working.

Of course many different factors can come into play when making this selection, but in my experience the two most relevant ones are project availability and organizational strategy.

Project Availability

It's likely that you already have at least one project in mind, but don't worry if you haven't. It's very likely that some good candidate projects are hiding in plain sight. Scout around the organization and see whether you can spot any of these:

- A tool that someone put together to make life easier for them and their team
- Resources created for customers (as long as they don't incorporate customer details)
- Projects from internal hackathons
- Unselected proofs-of-concept
- Products that have reached end-of-life

These are a few of the places you can look for projects to release. As the organization releases projects more often, this step will get a lot easier. People will start coming to you asking to release their project, and some projects may be built from the start with an intention to release them.

Typically, you'll end up with several decent options after having a good look around. If, after rummaging through the metaphorical sofa cushions, you still can't locate a good candidate project, you always have the option of creating one from the ground up. This can lead to some great projects, but it's also expensive. Is it worth investing all that time, talent, and treasure just to make something that the organization is going to give away for free? The answer to that relies entirely on whether the return on that investment is valuable enough for the business strategy.

Strategy

Have a good look at the goals from [Set Goals, on page 301](#). Now have a good look at the list of candidate projects that are available to you. For each of those candidates, answer these questions:

1. Will releasing this as FOSS support the goals?
2. If yes, how will it do so—which is to say, what value will it provide toward the goals?
3. How can you show or prove that?

When you're just starting out with releasing outbound FOSS projects, the best project candidates are those that not only further the business strategy but also are easy to see whether they're actually working toward that strategy. Metrics

such as new customer leads, increased conversion rates, or decreased employee turnover are all relatively easy to measure with modern technology. Others, such as customer sentiment and brand recognition, are more fungible and therefore also more difficult to collect and quantify. This is especially true when trying to use metrics of questionable validity, such as the popular Net Promoter Score (NPS).¹³

Start your FOSS releasing on easy mode; choose to release projects that make it easy to see whether they're making a difference.

What Business Are You In?

If you've gotten this far into the book, you have a firm grasp on how both inbound and outbound FOSS can help your business. In retrospect, it may even seem obvious that engaging with FOSS projects and communities furthers strategies and strengthens the overall business ecosystem. This is the way things have been operating for decades, whether companies have noticed it or not.

Despite that, I frequently come across companies with leaders who are reluctant to release any sort of materials under FOSS licenses. They cannot fathom how there might be value in giving away the company's hard work for free. "We created this tool to verify website accessibility compliance. We can productize it." Well, yes, you can. But should you?

When faced with thoughts or objections like this, always keep the business front and center. Remind yourself what business you're in. Are you in the website accessibility compliance business? Or are you in the financial services business? Because if it's the latter, then spinning up an entirely new line of business just to market website accessibility is going to be a pretty big pivot and probably won't make much sense for the company. On the other hand, if the project isn't a part of what differentiates your products from your competitors', releasing it as FOSS can bring all of those benefits that you learned about in [Why Organizations Release Projects, on page 298](#).

Netflix isn't in the chaos engineering business.^a Google isn't in the code repository management business.^b Mercedes-Benz isn't in the software security testing business.^c Be like these successful companies. Be fearless and share the projects your company builds that aren't directly a part of the value proposition it offers to its customers.

- a. <https://github.com/netflix/chaosmonkey>
- b. <https://opensource.google/projects/gerrit>
- c. <https://github.com/mercedes-benz/sechub>

13. <https://www.jstor.org/stable/30163980>

Release Non-Software

So often when we speak of FOSS, we're afflicted by software on the brain. Yes, that final S in FOSS stands for *software*, but as you learned in [Other Types of Licenses, on page 54](#), you don't need to limit your project releases to software only. As you'll see in the example [Bees4Less and the Open Hive, on page 313](#), Bees4Less released several Open Hive projects under open licenses—specifications, CAD files—but none of those projects were software.

Data, hardware, and creative works such as visual art, music, and writing can all be powerful tools in your free and open strategy tool chest. Each of these types also requires the same due diligence and preparation to release as software, so the information in this chapter applies equally to them.

Something to keep in mind for these types of projects, that isn't as large of a concern for software, is whatever you release should be compatible with open tools and formats. If potential community members need to pay hundreds of dollars for a tool like Adobe Photoshop or Autodesk CAD software to use the projects your company releases, you're limiting the potential of the project as well as the benefits your company can hope to see from releasing it.

Styles of Project Releases

As you'll see in [Chapter 20, Releasing FOSS Projects, on page 317](#), the basic steps to release a project are more or less the same regardless of what you're releasing. Those steps may differ in scope, and there may be more or fewer details to sort out, but the steps themselves don't change too much. The difference in project release style comes in what you do *after* the project is out in the world. You'll find many different styles, depending on how you decide to approach follow-up after releasing, but these are the four most common.

Throw It Over the Wall

When someone releases a project with no intentions or plans for engaging with it afterward, we say they *threw it over the wall*. The project is publicly available under a free or open license, but the originator of the project won't be providing maintenance, community management, responses to bug reports, or anything else. They've booted the project out of their space: out of sight, out of mind.

Companies use this style of release fairly often, and sometimes unintentionally. Unaware of the potential business value in a well-managed FOSS project, leadership agree to internal staff requests to release projects but are oblivious to any need for a budget or plan for ongoing support. Other times, they're aware of a possibility for that ongoing support but find it's not a strategic

priority to invest that sort of time, energy, and budget in it. They do, however, see value in the [Attention, on page 298](#), that a growing collection of released FOSS projects can gain them.

These projects are usually some sort of internal utility or resource for which the originating organization has no further need. Rather than it taking up space inside the organization's infrastructure, they throw it over the wall to a forge or other sharing service on the off chance someone else might eke out some additional benefit from it. Beyond perhaps some positive attention for sharing resources, the org has few to no expectations of return on investment or value generation from the project.

However, low expectations don't equate to no risk. Unless the originating company is very clear at the outset that it has no intentions of supporting or maintaining the project in any way, it can be seen as unreceptive and untrustworthy, damaging its reputation and brand. When deliberately throwing a project over the wall, declare engagement constraints in the project's README and CONTRIBUTING files. If possible in the selected forge, disable the ability to open issues or tickets as well as the ability to send fixes, patches, and enhancements. These steps will make it clear that the originating organization is washing its hands of the project and merely making it available in case someone else may find it useful.

Release but Retain Control

It's common for companies that are just getting started with releasing FOSS projects to retain as much control of the project as possible afterward. The entirety of the project's governance remains in the hands of the company, which also drives the roadmap and provides nearly all of the contributions to the project. They're not especially open to community contributions (and may even fear them) and not adequately staffed to provide community management.

Most often, this sort of release style happens when a company goes with what they know, making the project publicly available but still trying to operate it as though it were a private, internal project. People do this all the time when they try something new—falling back on something with which they're already familiar. For instance, if you already know how to play guitar, you'll try to apply those skills when you first pick up a banjo, but mastering a banjo requires a different, if related, skill set.

Some companies intentionally choose this release style for strategic reasons. Open core companies—which you'll learn more about in [Chapter 22, Incorporate FOSS into Business Models, on page 369](#)—regularly release business-critical projects as *single-vendor commercial FOSS*. The project

acts as a loss-leader, attracting people whom the company hopes they can convert into paying customers. Because these projects are the core of the chosen business model, open core companies often don't need or want outside contributions, as these may not align with their strategic direction.

Other companies will retain control over their released FOSS project only for as long as it takes them to build momentum, a user base, and a community for it. They see this control as a transitional step toward the project gaining open governance and community, which is more sustainable for both the project and for the organization that released it.

As always, when releasing a project and retaining control, it's critical that the originating organization set up appropriate community expectations from the very start. All project documentation should make the project governance clear and reinforce minimal expectations for community contributions. Ideally, the project would provide an alternative outlet for those outside of the company who wish to participate in some way, such as a chat or discussion board where the community can gather and help each other. Ignoring the strong human desire to lend a hand will alienate those who might otherwise become strong advocates for the project and perhaps the company.

Release with Open Governance and Community

FOSS projects with open governance and strong communities are the sweet spot. If an organization is able to release a project and develop this sort of community engagement, it'll be well placed to reap the most benefits from its outbound FOSS initiatives.

By necessity, when a company first releases a project with the intention of it having open governance, that governance will need to skew heavily toward company participants. There hasn't yet been the chance to build the community necessary to have a diversity of organizations represented in the governance structure. In this way, it may at first appear this style differs little from [Release but Retain Control, on page 309](#), but a company can explicitly signal its intentions for open governance at the moment of project release in a variety of ways:

- Ensure that from the start the project can stand on its own without relying upon company infrastructure.
- Prepare governance documentation for open selection of leaders, policies, and roadmap, and announce criteria and a timeline for that selection.
- Reach out to potential users and contributors in advance of the public project release so it can launch with some external stakeholders on day one.

This style of project release helps an organization develop or enrich an ecosystem, establishing it as an important partner and authentic collaborator in that space. By doing so, the org positions itself to take advantage of the possible benefits listed in [How Releasing FOSS Can Benefit Your Strategy, on page 249](#). Many of those benefits are possible without an openly governed FOSS project, but the benefits may not be as easy to gain.

Releasing a project with the intention of it standing on its own with open governance doesn't mean the organization also intends to walk away from it. They're not throwing it over the wall; they're collaborating. Usually when a company releases a project and its able to establish a strong community and open governance, the community and other contributors view the company as a "first among equals," affording it the respect earned by sharing the resource with the world. The originating company may always choose to withdraw from the community later if it's no longer a strategic priority, but it can do so confident that its creation will continue without its oversight.

Note, though, that building the strong and diverse community required to sustain a project under open governance takes a lot of attention and intention. It doesn't happen by chance, and it cannot happen in a toxic environment. If your organization hopes to release this style of FOSS project, make sure you budget and plan for adequate community management and participation right from the start. Failure to do so may mean the project never gains the momentum necessary to get off the ground and build the community necessary to stand on its own.

Donate to a FOSS Foundation

Donating a project to a FOSS foundation is a more formalized style of an open governance and community release. The foundation provides a neutral and supportive playing field for FOSS contributions. Those contributions typically occur according to the foundation's established infrastructure, framework, or workflow. The foundation also manages the legal aspects of the project, including trademarks and any copyright or licensing matters. Because the foundation holds the project in its neutral space, it makes it easier for competitors to come together and collaborate for coopetition.

Many different foundations provide administrative homes for FOSS projects.¹⁴ By far the largest of these is the Linux Foundation,¹⁵ which includes under its umbrella dozens of smaller subfoundations. It may be the 800-pound

14. <https://flossfoundations.org/foundation-directory/>

15. <https://www.linuxfoundation.org>

gorilla of FOSS foundations, but it's far from the only one. It also may not be the best fit for the project your organization intends to release. You'll learn more about selecting the right foundation for a project in [Selecting a Foundation, on page 342](#).

Organizations that donate their projects to foundations wish to remove any perception that they “own” or control the projects. They participate in the continued project development and may hold a leadership position within the project governance structure, but ostensibly they're no longer responsible for the health and sustainability of the project. That responsibility falls to the foundation.

In reality though, most foundations won't accept projects unless the originating organization is a supporting member of the foundation. Corporate membership almost always comes with a price tag, and sometimes that cost is considerable (up to hundreds of thousands of U.S. dollars). Keep this in mind when the question of whether to donate a project to a foundation arises inside your organization.

If All Your Friends Jumped off a Bridge, Would You Follow?

At this moment it's somewhat trendy for companies to release FOSS projects. As with any trend, this one also comes with peer pressure. An “if you're not releasing FOSS, you're doing business wrong” sort of thing.

Don't listen to those people telling you to jump off the FOSS bridge with them. Whether and what to publicly release under FOSS licenses is a *business* decision, and like all decisions the answer may very well turn out to be “nope, not gonna do it.”

Releasing FOSS projects may not be the right strategic decision for your organization for many reasons:

- You may not have enough resources (time, talent, treasure) to do it well.
- You may not have projects you feel comfortable releasing.
- Any number of other unique-to-your-firm reasons not to release FOSS may be in play.

As a FOSS advocate, of course I would like more releases under free and open licenses, but as a business strategist I also want companies to remain solvent and allow their employees to keep roofs over their families' heads. As a FOSS advocate and business strategist, I'm here to tell you that *it is OK not to release FOSS projects*.

If it's strategically good for the firm to release projects (and it often is), then by all means do so. But if the situation isn't currently right to release that project, then just hold onto it for now. As you learned in [Chapter 18, Develop Your Contribution Strategies, on page 283](#), you can always find other ways to support critical FOSS projects while also helping your business reach its goals.

These are the most popular styles of FOSS project releases, and they're fluid. Your organization may throw a project over the wall, then decide later to invest in developing a community under open governance. Or it may release a project under tight control, then later donate it to a foundation. Or it may release a project under open governance, then abandon it (effectively throw it over the wall) when internal political support for it wanes.

While the transition between these styles may not always go smoothly (often due to lack of communication internally and with the community), that transition is still usually possible. These styles aren't dogma, only guidelines. Take from them what your organization needs to support its strategic goals.

Now that you have a better idea of some of your release options, you're ready to start working on the release plan. But first, let's return to our friends at Bees4Less to see how this could work in practice.

Bees4Less and the Open Hive

Bees4Less has developed a new standard for interchangeable parts for moveable-frame bee hives. This standard has increased hive quality while reducing production costs for Bees4Less, so they like it a lot. The apiarists find that it's a lot easier for them to maintain hives that have standard-based components, so *they* like it a lot too. Bees4Less figures that if they can promote the standard and get it enough adoption, the entire industry will adopt it. They'll have first-mover advantage plus the goodwill that comes from developing such a useful standard.

That's their goal: increase adoption of the new hive component standard.

Challenges

Marketing and Sales report that the new design practically sells itself when a customer is looking for all-new hives. Once they know that the product exists, customers see the value in the design, giving them a high willingness to purchase and a low sensitivity to price.

But it's proving difficult to get people with existing hives—theirs or their competitors'—to even look at the new and improved design. The teams are having a hard time getting their attention and are looking for ways to establish trust and brand recognition.

Launch of the Open Hive Foundation and Standard Specification

The Bees4Less team make their move once the new line of hives has launched and is selling pretty well: they introduce the Open Hive Foundation (OHF) to the world.¹⁶

OHF is a 501(c)(6), member-supported, nonprofit organization based in Oregon in the United States. Its purpose is to maintain and promote the interests of the Open Hive Standard (OHS). Bees4Less donated the standard specification to the foundation, releasing it under a Creative Commons Attribution-Share-Alike (CC BY-SA) license.¹⁷ OHF also holds and maintains any OHS and OHF-related trademarks.

Prior to releasing the standard, Bees4Less acquired the necessary patents on the innovation. It donates these to OHF, which uses them as seeds for an Open Hive patent pool. Companies that join OHF as members may use these patents freely and openly.

While Bees4Less created the Open Hive Foundation and provided the initial funding to get it started, it operates as an independent entity with its own open governance. Its ongoing funding comes from membership dues and licensing the patents to those who don't wish to become full members of the foundation. Bees4Less is an inaugural member and has a seat on the Board of Directors. They recruit a few other companies—including a competitor—to join as inaugural members, largely on strength of access to the patents.

The OHF members come together not only to use the patents but also to develop the standards and programs to promote and support it. One of OHF's first tasks for them is to design and launch an Open Hive Certified™ program.

Release of CAD Designs for Components

Bees4Less already has computer-assisted design (CAD) files for all their hive components. They then buy three of the top competing models of hives and—after confirming with the legal team that it doesn't violate IP laws—they develop CAD files for adapters between the competitors' components and Bees4Less's new standard ones. They then release these CADs under the CERN Open Hardware License Version 2—Strongly

16. <https://openhive.foundation>

17. <https://creativecommons.org/licenses/by-sa/4.0/deed.en>

Reciprocal (CERN-OHL-S-2.0) license.¹⁸ At the same time, they start selling these adapters as a part of their Open Hive product line.

Consider the Risks

When coming up with this plan, the Bees4Less leadership considered the ways it could all go horribly wrong for them.

Their competitors could take the specification and create their own hives according to it. Wouldn't that undercut the Bees4Less sales? Possibly, but to do that the competitors would first need to gain access to the patents, then design their hives, and finally tool up for manufacturing. All of this would take a lot of time, during which Bees4Less would continue to bolster its reputation for quality and customer service. By the time the competitors catch up with their own hive products, Bees4Less will have moved on to the next stage of their business strategy.

The specification is openly available for anyone to see, use, and modify. Will the customers take this and build their own hives instead of buying from Bees4Less. Some of them may, but these are the people who already would prefer to build rather than buy. Therefore, these folks were unlikely ever to buy from Bees4Less. By enticing them to engage with the standard though, they've successfully gotten these people to join the community. This increases the Bees4Less brand recognition and reputation.

Those CADs may allow anyone to create their own replacement components instead of buy them. Won't that be a problem? Probably not, since these people also are the sort to prefer to build rather than buy and therefore weren't in the Bees4Less target market. Similar to the hive-DIYers, this group of people probably wouldn't have otherwise joined the community, but here they are. That's a net win for adoption of the standard in the industry.

Results

An awful lot's going on here, and it's all good for Bees4Less. They've developed a standard that reduces costs while increasing quality and value. They also have a very strong first-mover advantage. By releasing the standard and launching OHF to hold and maintain the related intellectual property, they've created compelling incentives for others in the industry to come together for productive cooperation while also promoting the standard. Let's not overlook the fact that by creating a product that's more standardized and modular, Bees4Less is reducing waste and upholding their company values of sustainability.

18. https://ohwr.org/cern_ohl_s_v2.txt

These strategic moves took a lot of planning to get right, but the end result is an innovative brand with immense goodwill from the market. Bees4Less is going to sell a *lot* of hives and pollinator-related products.

Now that you've seen how this might look in a real-world application, it's time for you to do it yourself. Let's learn about how to release a FOSS project.

Releasing FOSS Projects

You're nearly ready to release a outbound FOSS project! You have only a few steps to go:

1. Collect the release team.
2. Choose a release style.
3. Name the project.
4. Prepare the project for release.
5. Decide whether to use a CLA/DCO.
6. Choose a release location.
7. Select a license.
8. Apply the license.
9. Comply with component licenses.
10. Verify P&P.
11. Release!

Don't be discouraged by the number of steps. After all, making a good sandwich takes a lot of steps too. Like that good sandwich, these steps also aren't too difficult and are worth the time to do right.

You'll learn about each of these steps and gain enough information to make the right decisions. However, as with so many things that are worth doing, a lot of additional nuances can take your work to the next level.

That amount of detail is out of scope for this book, but thankfully someone else has already written a superb resource that will give you a boost up to that vaunted next level: *Producing Open Source Software* by Karl Fogel.¹ It was the first book to detail how to release and support FOSS projects, and

1. <https://producingoss.com>

it's still the best. The most recent version is available for free online under the Creative Commons Attribution-ShareAlike (4.0) license.²

Collect the Release Team

Never underestimate the importance of having a team to collaborate on this effort. None of us have all the skills and time required to do everything *and* to do it well, and we all need help, feedback, and sounding boards. Assembling a team at the very start of the process will ensure everyone has the information required to make this project release a success.

Team Composition

The release team should comprise representatives from all relevant departments within the organization. For instance, if in [Choose What to Release, on page 305](#), your company decided to release a software project, some departments that may need a seat on the release team can include these:

- OSPO
- Programming
- Technical Writing/Documentation
- Marketing
- Security
- Legal

Take a good look around to determine what departments in the company may be a stakeholder in or may otherwise be impacted by this effort. For instance, if the project you intend to release is relevant to the company's product offerings, you may wish to include representatives from Product, Sales, and Customer Service.

Remember: team composition isn't carved in stone. You're free to bring in new team members at any time should it be necessary. So when the team learns that the release process will require a change to access through the corporate firewall, don't hesitate; bring an IT representative onto the team to identify and help with this and any other related matters.

Team Operation

It's not necessary for every team member to be hands-on for each step in the release process. Usually, it's enough for many of them to be in the loop and aware of what the other teams are working on. For example, Marketing may

2. <https://creativecommons.org/licenses/by-sa/4.0/>

not need to be involved with security reviews, but being aware that reviews are happening may provide information that Marketing can use in their messaging.

Like all efforts, this one benefits from a diversity of perspectives, experiences, and opinions. A varied team composed of representatives from across the company provides this. Each team member should be empowered and encouraged to ask questions and provide feedback at each step in the process, especially if it's an area in which they're not an expert. What some may see as “dumb” questions with “obvious” answers often uncovers assumptions that could make life difficult later on. Creating a blameless and psychologically safe space for team members to question and provide feedback leads to a better project and a smoother release process.³

Team Leadership

We've all had to suffer through committees and teams that appear incapable of delivering. Often this is due to weak, poor, or even no leadership. Don't let your release team get caught in this trap; ensure it has a skilled and empowered leader at its helm.

Often this leader comes from the organization's OSPO, but they could be a representative of any of the stakeholder departments. They must seek out and accept feedback from all team members and work to build consensus but be empowered to make a final decision should consensus not prove possible and the effort is at risk of stalling. This decision-making responsibility requires that the leader either be the executive sponsor of the effort or that they have the trust and support of that sponsor. Following are some other useful skills for this leader:

- Project/program management
- Communication
- Office politics
- Diplomacy
- Conflict resolution

While team lead is a high-profile role that bears a lot of responsibility, it's also one that requires a great deal of time and effort. Do your best to be realistic with selection of the leader. The best person for the job may already be overloaded with tasks. Make sure the person selected to lead the team has the capacity to do the work necessary to keep the release effort on track.

3. https://en.wikipedia.org/wiki/Psychological_safety

Choose a Release Style

In [Styles of Project Releases, on page 308](#), you learned about the four most common styles of FOSS project releases:

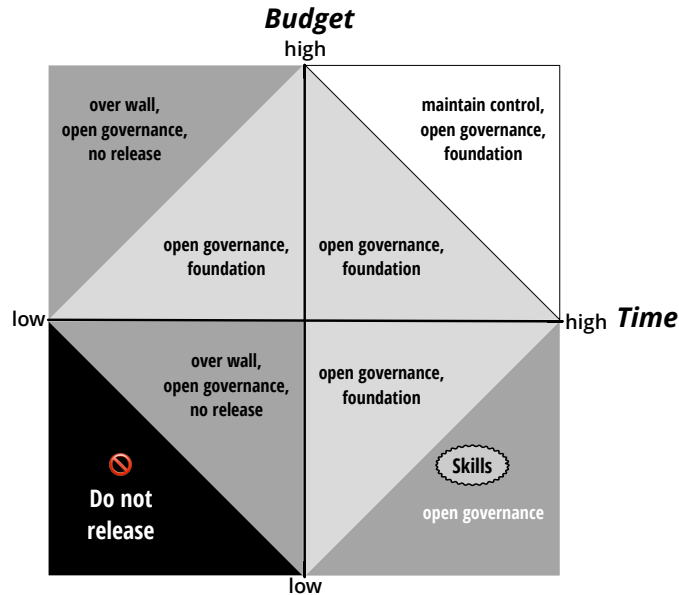
- Throw it over the wall
- Release but retain control
- Release with open governance and community
- Donate to a FOSS foundation

Whereas the question of whether to release a project at all largely relies on the constraints of strategy alignment and leadership support, the question of which release style to choose leans strongly on three other constraints: budget, time, and skills. Let's use a 2x2 again, but this time it'll be a bit more complex. You can find the 2x2 diagram in the appendices at [Choose a Release Style, on page 422](#). Here's how you can use it to help you decide which release style to use:

1. From the three constraints—budget, time, skills—assign one to the X-axis and one to the Y-axis. Choose the two that you feel are the most important or constrained in your organization since they'll be the most limiting factors of your plan. It shouldn't matter which one you choose to put on which axis. In the example that follows, I chose to assign Budget to the Y-axis and Time to the X-axis.
2. Make your informed guess as to where along the high-low spectrum each of these constraints falls. This will give you a quadrant on the 2x2, reducing the number of release styles you need to consider. In the example, a medium-to-low budget is available but medium-to-high time. This puts it in the lower-right quadrant, which limits the style possibilities to open governance and foundation. This may be enough for your purposes. If so, you can stop there and research the options listed in the quadrant.
3. If you need more help, you can consider one final constraint (in this example it's Skills). Knowing what you do about that constraint, where do you feel it should be placed in the selected quadrant? That can help you limit your style options down further if need be.

Yes, it's a feeling for that final step. This isn't a science. It's a tool to aid decisions and discussions, not the decision itself. Don't mistake the map for the terrain.

Once you've plotted your constraints, it should look something like this:



If you're not keen on the styles found in the quadrant in the second step, you're not required to use one of them. However, you should keep these style/constraint correlations in mind:

- *Over the wall:* Usually requires the least budget, time, and skill, but it also provides the fewest benefits.
- *Retain control:* Because of its high-touch nature, this style requires the most of all three of these constraints.
- *Open governance:* The assistance of the community can make up for a lot of constraint limitations, but the style still requires at least some time and skill to initiate (or budget to pay a skilled external person to handle it).
- *Foundation:* The foundation can handle a lot of the details, freeing up constraints internally, but you still need people to stay involved and also may need to pay for foundation membership.

Could Is Not the Same as Should

“...your scientists were so preoccupied with whether they could that they didn't stop to think if they should.”

—Ian Malcom, *Jurassic Park* by Michael Crichton

As Doctor Malcom demonstrates here, “Could we?” and “Should we?” are two entirely different questions and must be addressed as such. [Release Strategically, on page 301](#), can help you to work through both questions, but you still may want to sit down and consider the answer to each of them individually.

As a race, humanity is eternally and boundlessly optimistic—often to our own detriment. As you work through this process, make sure you're doing so with your eyes wide open and with realistic expectations. Perhaps now isn't actually the right time to release a project. That's OK! You can always do it later.

Maybe, when you're honest with yourself about it, you realize that the company doesn't *actually* have either the skills or the time necessary to execute the plan well. That's also OK! You can pause, then take the time to train people, hire outside help, or both. A lot of problems can be solved by throwing money at them, after all.

What's not easy to do is untangle the mess created when a neglected Could meets an ignored Should. This is the sort of mistake that can have immense costs, both financially and politically. Save yourself the headache and make sure you have honest answers to these two critical questions before you go much further in the process.

Name the Project

One of the fundamental truths of software is *Naming things is hard*. You'll find that some of the most animated conversations you have during the release process may be about what to name the project, software or not. Everyone will have an opinion, and more people than you'd imagine will want to share theirs.

The final name for the project is something that's highly specific to your organization, but here are some tips you can keep in mind that may help you throughout the (potentially contentious) naming process.

Consider Following Existing Conventions

If the project will be a part of an existing ecosystem, have a look at some of the other projects there to see whether their names follow some sort of pattern or convention. For instance, because the name of the project means *helmsman* in Ancient Greek, projects in the Kubernetes⁴ ecosystem often have nautical

4. <https://kubernetes.io>

names, logos, or both. Some ecosystems commonly use a prefix, such as *Py-* for Python projects, *WP-* for WordPress, or *go-* for the Go programming language.

Following existing conventions isn't required but can make life easier for the naming team now and for users later. The naming team can use this method to limit the possibilities they need to consider. Later on, potential users can get a sense of which ecosystem the project is a part of and therefore whether it's a good fit for their needs.

Self-Documenting Names

Similarly, it is helpful to potential users and community members if the project name somehow reflects the project purpose. In programming, this sort of name is known as *self-documenting*: it makes it clear what the thing does, and this clarity avoids confusion.

For example, the financial services company from [What Business Are You In?, on page 307](#), could very easily name its accessibility compliance tool Hyacinth. It's a perfectly good name and a lovely flower. As a potential user you may hear people talking about Hyacinth, have no clue what it does, and feel it's more trouble than its worth to learn more. Naming the project *go-accessible*, on the other hand, provides information; it tells the reader that the project is written in Go and that it's related to accessibility. It may not be a lot of information, but it's a heck of a lot more than a reader gets from Hyacinth. They can tell at a glance whether they'd like to give the project more of their attention.

Again, this is only a guideline for project naming, not a rule. Hyacinth may have many valid arguments in its favor. If that's the case, of course it can be a fine project name. However, should the name not be self-documenting, plan around this shortcoming by providing additional information in all documentation and project resources so it's easy for people to determine what exactly the project does.

Avoid Including the Company's Name

It can be so tempting to include the company name in that of the project. Every time someone uses the project, the firm gets that sweet, sweet brand recognition. Surely that's a great idea, right? Wrong. Many times, having a project named after the organization (or vice versa) leads to frustration and headaches.

One problem it causes is that of community perception. When external people see a name like Bees4Less Open Hive, they're justified in assuming that the project is by and for Bees4Less only. This can dissuade them from joining the community, as they may feel that they'll have little opportunity to provide input to the project. Potential contributors may be reluctant to share their time and expertise with the project, feeling like they'd be providing free work to a for-profit firm.

Another problem is the business complication that comes from having a FOSS project nominally entwined with a for-profit entity. Rather than boosting brand recognition, it's possible that the collision of the names instead causes brand *dilution*, such that people are more likely to think of the project rather than the company.

Then there's the matter of how to disentangle things in the case of a merger or acquisition (M&A). If the company is acquired, what happens to the project of the same name? It's under a free and open license and therefore freely available to everyone. Relicensing under a proprietary license is an option but won't preclude people from forking the FOSS version of the project (see [Do You Have the Spoons to Deal with a Fork?, on page 45](#) to refresh your forking memory).

This brings up another problem: trademarks. The name of the company is undoubtedly registered as a trademark in the appropriate jurisdictions, so the project name includes a trademarked term. This can cause unnecessary complications. For instance, what happens in the case of a merger/acquisition? Here are some things to consider:

- How will an M&A affect the use of that trademark by the project? Regardless of M&A status, the community always has the right to object to the running of a project and fork it in reaction.
- Even if renamed, that fork will retain vestiges of the trademark. Will this be a problem for your firm? Will it take legal or other action over this use of its trademark? An undefended trademark is a vulnerable trademark, so some sort of action may be required.
- Finally, having that trademark in the name will assuredly preclude donating the project as-is to a foundation.

Despite decades of examples showing that these problems aren't theoretical, companies continue making the mistake of sharing some or all of the name

of the project. It's an unforced error,⁵ and it's easily avoided: don't include the company name in the name of the project. The organization has plenty of other ways to gain recognition from its release of and participation in the project. For example, consider these:

- The name of the organization could be in every copyright statement in every file of the project (you'll learn more about this in [Add Copyright and License Statement to the Work, on page 350](#)).
- The project documentation can name and link directly to the company and even include the company's logo.
- The organization can sponsor project-related events and materials, ensuring visibility for its name and logo.

The reasons not to include the name of the organization far outweigh any for including it. Make life easy for yourself and avoid these problems even before releasing the project.

Consider Project Branding or Namespace

If your organization either already has or plans to release several projects around a similar topic area, consider giving them all similar names or having them use a shared namespace. This helps to aggregate the branding for the project as well as increase the visibility for all projects under that single namespace umbrella.

Netflix did a great job of this when they released what came to be known as the (now retired) Simian Army.⁶ Each of the projects in the Simian Army namespace had monkey- or ape-related names, such as Chaos Monkey, Chaos Kong, and Security Monkey. People who operate in the chaos or resilience engineering spaces knew what to expect when they came across a new project in this namespace, which developed a strong brand for usefulness and reliability.

Choose an Inclusive Name

Humans have developed thousands of languages, containing billions of words that can be arranged in infinite ways. Despite the boundless options available, some people insist it's perfectly OK to choose names that perpetuate marginalization in our society and cultures. Those people are wrong.

5. <https://www.merriam-webster.com/dictionary/unforced%20error>

6. <https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116>

Terminology or imagery that disregards or disrespects other people, cultures, and experiences is easily avoided. Doing so is almost literally the least a project can do to show it welcomes use by and contributions from others. As the naming team is brainstorming, take a few moments to review candidate names to ensure they're respectful. Here are some tips to help with that process:

- Don't use names related to current or past religions or cosmologies. Naming a project *Oyin* (the Yoruba name for bee) is probably safe, but naming it *Ọlọrun* (the creator of the universe in the Yoruba religion) could be a problem.
- Don't use imagery based on other cultures and their practices. Many sports teams in the United States are finally shedding their disrespectful names and logos related to Native American tribes and cultures.
- Check resources such as the Inclusive Naming Initiative,⁷ the American Psychological Association Inclusive Language Guide,⁸ and the Responsible Communication Style Guide⁹ to make sure the name candidate isn't inadvertently offensive to marginalized communities.

The effort spent doing this inclusivity verification will be a tiny fraction of all the effort required to determine and apply a name to the project and will pay immense dividends by signaling to users and community members alike that they are welcome in the project.

Look for Name Collision

Millions of FOSS projects are available today, and millions more will be released in the coming years. With that number of projects extant, it's inevitable some of them will share the same name. Inevitable or not, you still should do a quick search to make sure your name candidates aren't already used by projects or organizations. If nothing else, doing so will allow you to avoid duplicating a name that may have been trademarked as well as any subsequent legal complications.

A web search is one way to check for a name collision, but it can turn up a lot of false positive results. Instead, a quick search on each of these sites will limit the results to FOSS or publicly available projects:

- GitHub¹⁰
- GitLab¹¹

7. <https://inclusivenaming.org>

8. <https://www.apa.org/about/apa/equity-diversity-inclusion/language-guidelines>

9. <https://rcstyleguide.com>

10. <https://github.com/search>

11. <https://gitlab.com/explore/projects>

- The Apache Software Foundation (ASF)¹²
- The Eclipse Foundation (EF)¹³
- The Linux Foundation (LF)¹⁴

Projects located at ASF, EF, and LF will have their names trademarked already. For projects located at GitHub or GitLab, verify trademark status before considering a name that may cause a collision.

Don't forget, project names are used for many things. As you're checking for collisions, make sure to check whether the relevant internet domain names and social media handles are already in use by other projects or organizations. Regardless, if you have even the hint of a question about whether a name collision might be a problem, consult with Legal.

Aid Discoverability

For most people today, their search engine is their primary gateway to the wealth of content available on the internet. Keep this in mind as you consider names for the project: is it unique enough to appear in the first page of search results?

Here are a few popular FOSS projects:

- Go
- Rust
- Compose
- Hugo
- Helm

Put any one of those common English words into your favorite search engine and it might take you a while to locate the actual project. Those in the know realize that to have the project in the first page of search results, they need to modify each of those search terms:

- Golang
- Rustlang
- Docker compose
- Hugo website
- Helm chart

12. <https://projects.apache.org/projects.html>

13. <https://projects.eclipse.org>

14. <https://www.linuxfoundation.org/projects>

Consider the target audience for your project. Are they already in the know? If it's a new project, they're probably not. So they won't have the knowledge required to modify a search term to locate your project if it has a name that's a common word like the preceding ones. Choose a name that minimizes the need for that modification. It doesn't need to be unique, only distinctive.

Do a Giggle Test

We all like a good joke, but we don't like to *be* the joke. Once you have a short list of name candidates, do yourself a favor and put that list in front of a blunt thirteen-year-old, someone who's tapped into the online community, or anyone else who has a completely different perspective from the naming team. If a name candidate makes that person giggle, discard it.

Even seemingly innocuous words can trigger an impish giggle. For instance, to you, Toparse is a perfectly logical and descriptive name for a project that parses the output of the Linux `top` command: `top` + `parse` = `toparse`.¹⁵ To your young cousin (and, frankly, to me), that name reads as “top arse” and is absolutely hilarious.

This short (and often fun) exercise does wonders for avoiding potentially embarrassing project names, words that you think have one meaning but may mean something completely different in current slang.

Should You Trademark the Name?

As you learned in [Trademarks, on page 19](#), in some jurisdictions you gain some legal protections simply by using the trademark, even without going through the effort of registering it. For smaller projects, that minimal amount of protection may be all they need. Also, projects released in the “throw it over the wall” project style are unlikely to need much or anything in the area of trademark protections.

Larger projects, projects that are related to the name of your organization or your products, or projects that are under foundations all are good candidates for trademarked names. Large project or small, consult with your legal team to see what they have to say on the matter of trademark.

If you do decide to trademark the project name, you'll also need to create and publish a trademark policy so people know whether and how they may use any trademarked material associated with the project. You can see many examples of trademark policies at the FOSS Governance Collection.^a

a. <https://foss-governance.org>

15. <https://man7.org/linux/man-pages/man1/top.1.html>

Prepare the Project for Release

The next step in the release process is to tidy everything up and make sure the project is ready for the public to see and use. This isn't a difficult step, but it includes several moving parts. The good news is that the release team can work on most of these parts simultaneously, which speeds up the process. The bad news is that none of these moving parts is optional, and it can be easy to forget or overlook one of them.

Set the Name in Stone

Unless the project is purpose-built to release as FOSS, it'll already have some sort of internal name. If that's not the same as the name chosen in [Name the Project, on page 322](#), then instances of the old name need to be replaced with the new one. Sometimes this can be as easy as a find-and-replace operation, but usually it's a more complex procedure.

For instance, if the project is software, then the old name may appear in the API, in variable names, or in code comments. Depending upon the scope of the changes, the release team may decide only to update the name on any user-facing elements but for the sake of prerelease stability save cleanup of old name instances in the source code itself as a project for later.

Sanitize and Secure the Project

Even the best development processes have cracks through which both minor and major potential shames can creep. Before you release anything to the public—product or project—it's always smart to give it a once-over to make sure it will reflect well on the company, correcting instances of potential embarrassment before they have the opportunity to damage the reputation of the organization.

Check for Trademark Usage

Have the team look through the project for any mentions of other companies, components, or anything else that might have a trademark on it. Do the following for each mention they find:

- Is the mention necessary? If not, remove or replace it. Most often, this is this correct action.
- If the mention is necessary, verify either that it adheres to the use policy for that trademark or that it's fair use (see [What's up with Fair Use?, on page 42](#)). Document this verification within the project itself so others will see the due diligence.

- If it's not either fair use or within the use policy, it's unlikely actually to be necessary. Remove or replace the mention.

This verification usually takes little time, but that small investment can pay large dividends. All it takes is one mocking or critical mention of a trademark today for everyone to have a bad day after releasing the project.

Verify Project Security

Ideally—and especially if the project is a form of software—the team will have involved Security early in the development and release project (revisit [Collect the Release Team, on page 318](#), for more information). If that hasn't happened yet, do it now. The security team must review the project to ensure it's safe to release. They'll undoubtedly already have their own prerelease checklist for things they need to verify. This list will include things like the following:

- Verify the project doesn't include sensitive information, such as authorization credentials.
- Determine the Supply-chain Level for Software Artifacts (SLSA).¹⁶
- Perform fuzzing¹⁷ and other forms of testing to spot and correct any security holes.

Depending upon the workload and checklist of the security team, this step could take a while. This is one of the many reasons throughout this book I've advised getting Security involved as early as possible. The sooner Security is a part of the process, the more everyone will be able to trust the project is secure.

Check for Patentable Material

Some companies and organizations value their patent portfolios highly. These firms may use patent licensing fees as an important part of their revenue model, or they may patent their inventions defensively to support their business strategy.

Whatever the reason, if it's important to your company to maintain and grow its patent portfolio, then you should make sure the project receives a patent review prior to release. It's likely that this review is a regular step in the development process, but mistakes happen, and steps fall through the cracks all the time. Work with Legal to verify the patent status of the project.

16. <https://slsa.dev>

17. <https://en.wikipedia.org/wiki/Fuzzing>

Clean Up Project Language

Look, we're all human. We all need to blow off steam once in a while, and sometimes that steam takes the form of profanity. For instance, the search for the fairly innocuous word *damn* in GitHub returns more than four million results.¹⁸ Other less socially acceptable English profanities similarly have million-plus search results. How would your company's leadership feel if the company's projects were found in that result list? Probably not too happy.

Plenty of words may cause offense. The *Bad Words* list by Luis von Ahn from Carnegie Mellon University includes nearly 1,400 entries, and that's only for the English language.¹⁹ George Carlin famously declared there are "seven dirty words" that can never be on television.²⁰ However your organization chooses to define profanity, perform a search across the project and tidy up any profane terms that may have slipped in somehow.

As you're looking for profanities, don't forget to look for other potentially offensive language:

- Unnecessarily gendered terminology or instructions
- References to other cultures or religions
- Racially charged terminology

The Inclusive Naming Initiative²¹ and the American Psychological Association Inclusive Language Guide²² are helpful resources here.

Offense is, of course, in the eye of the beholder. What may be offensive to you may be entirely normal to me. What's important isn't whether *you* agree with a term's profane status but whether *anyone* does. Why risk offending others when there are so many alternative ways to phrase something? It doesn't take much work to be inclusive of others, and it pays off with a more welcoming project and vibrant community. Make the small effort; clean up the language.

Perform Other Internal Checks

Many organizations use an internally developed product launch plan or checklist. This plan ensures they don't miss any vital steps in the otherwise hectic task of releasing a new product into the world.

18. <https://github.com/search?q=damn+&type=code>

19. <https://www.cs.cmu.edu/~biglou/resources/bad-words.txt>

20. https://en.wikipedia.org/wiki/Seven_dirty_words

21. <https://inclusivenaming.org>

22. <https://www.apa.org/about/apa/equity-diversity-inclusion/language-guidelines>

If your company has one of these plans or checklists, have a look at it now. Does it include any steps that make sense to apply to releasing this FOSS project? It's likely that it does. For instance, depending upon the project being released, it may not make sense to prepare a demo for the sales team, but it may to write a press release.

You won't know for sure what steps may apply to this project release unless you review the product launch plan, so take a few minutes and do that before the project release process goes much further.

Is the Project Ready to Release?

Short answer: no. The project is not ready, and it never will be. Release it anyway.

So many companies and organizations hold back from releasing projects until they're perfect. This is a mistake. A perfect project doesn't exist; it's only ever "good enough." The mantra you want to remember is "Release early, release often." Don't allow a mythical perfection to become the enemy of the very real good enough.

Many businesses aren't comfortable with this sort of uncertainty, especially those in more risk-averse industries, such as finance and healthcare. For these, it can help to reframe the positioning of the project, categorizing it in a way that calms any rattled nerves. The FOSS nonprofit OW2 has created and published Market Readiness Levels that can help with this reframing.^a

Using the OW2 terminology and method, it's possible to create a FOSS release rationale that uses terminology that may be easier for risk-averse executives to understand and accept.

a. <https://www.ow2.org/view/MRL/>

Lay Governance Groundwork

Our society requires infrastructure to operate: roads, bridges, sewers, power transmission. Similarly, software requires a different sort of infrastructure: servers, memory, operating systems. Then there's *governance*, which is the infrastructure for human operation, in this case within a FOSS project.

FOSS governance includes anything that might express ways a project and community operates. This can include the project mission and definition of important terms such as "contributor" or "core team member," among many other subjects.

Some projects follow established models for governance,²³ but more often a FOSS project's governance is an emergent property of the norms and values of

23. <https://www.redhat.com/en/blog/understanding-open-source-governance-models>

the community that coalesces around it. This is a good thing! Project governance is something that should evolve along with the needs of the project and its community, not be inflexible, as though it were carved in stone. However, that doesn't mean a project should start with no governance guidance whatsoever in the hopes that a healthy method of governance will grow over time.

As your organization prepares to release this project, consider and document some important governance matters that will help keep the project stable as it's getting started in the world. Some of these matters include the following:

- The decision-making and tracking process
- Project and program management flow²⁴
- Security policies and contact points
- Leadership structure and selection

It's tempting to try to define All Things Governance right up front, with the good—albeit misplaced—intentions of removing that complication for the project in the future. Try to avoid this. Set up just enough governance to allow the project and community to stand on its own while still affording it plenty of room to evolve and grow.

As you're establishing that minimum viable governance, reference existing examples to see how other projects in the past have done this rather than trying to reinvent the wheel. The FOSS Governance Collection is a searchable and growing catalog of governance documents from dozens of FOSS projects of all types.²⁵ Use this resource and web searches any time the project team finds itself wondering, "How do other projects handle this?"

One of the key values of free and open source software communities is *transparency*. Therefore, document all governance matters and ensure these documents are publicly available. At any time, anyone should be able to locate, reference, and suggest changes for any element of the project governance structure.

Document Document Document

Seventy-two percent of surveyed software developers say that good documentation is a factor in deciding which FOSS project to use²⁶ and 93 percent say that bad documentation is a big problem in FOSS.²⁷ Despite the proven importance of documentation for the success of a FOSS project, docs—like

24. <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>

25. <https://foss-governance.org>

26. <https://tidelift.com/subscription/managed-open-source-survey>

27. <https://opensourcesurvey.org/2017/>

security—are unfortunately often left until the very end of any project and are therefore rushed and incomplete.

Governance is only one of the many topics that will need documentation for the project to have a good chance of success. Here are some of the other docs that just about every FOSS project needs:

- *README*: The front door of the project, and the first thing people usually see. It should provide the what, why, and where for all things in the project, linking to all other relevant resources. The best README files are like a one-stop shop where maintainers, contributors, and users can find pointers to the information they need to get stuff done.
- *Installation, configuration, and setup*: Whether it's consumers trying to use the project or contributors looking to set up a development environment, people need to know how to acquire and set up the project resources.
- *Usage*: Acquiring a tool is one thing; learning how to use it is another completely. Many people won't invest their precious time in puzzling out how to operate an undocumented project (this includes a project's API).
- *Contribution guidelines*: Unless the selected project style is [Throw It Over the Wall, on page 308](#), the project will need to tell people the process they should follow to make contributions. If this isn't documented, people will still try to contribute, but they'll make up their own processes for doing so. I can guarantee that this will waste a lot of project time, so document it now to avoid irritation on both sides of the contribution relationship.
- *SECURITY.md*: Especially for software projects, it's important to let people know where and how they can report potential security vulnerabilities. The standard method for doing this is in the SECURITY.md file. Before coming up with your own reporting method, check the documentation for the forge you selected. It may already have a preferred set of tools or a workflow that you can use or build upon.
- *Code of conduct and related processes*: Codes of conduct (CoC) exist to establish and maintain standards of behavior, codifying the “how we treat each other” for the project, and they are absolutely required for building and maintaining healthy communities. After all, it's unreasonable to hold people to unexpressed rules or expectations. A good CoC always includes information not only on how to report problems but also on how those reports are handled.

You'll find detailed guidance for all these documents and more in *Producing Open Source Software* by Karl Fogel,²⁸ *Program Management for Open Source Projects* by Ben Cotton,²⁹ and in my first book, *Forge Your Future with Open Source*.³⁰ Several other good resources are available online. I recommend *The Good Docs Project* as especially useful for FOSS project creators.³¹

If your organization has chosen to throw the project over the wall, releasing it without any expectations of future maintenance or management, declare that in the README file, contribution guidelines, governance, and anywhere else people may look for information in the project. Be clear, and establish these expectations with potential users and contributors so they don't get frustrated when their questions and contributions are ignored.

A Love Letter to Documentation

As you learned in the preceding section, documentation is important to software developers. Like many things that are valuable, it takes a lot of time and effort to write good documentation. This is time that most FOSS maintainers don't have, such that 41 percent of surveyed maintainers say they have trouble keeping up with necessary documentation.^a

The best way to make your project stand out from the crowd is to reduce this documentation bottleneck by hiring at least one dedicated technical writer for it and related outbound FOSS engagements. That one person will make an outsized positive difference in how others perceive the project and therefore in project adoption.

How can documentation make such a big difference? For one thing, docs are an amplifier of information: write them once and share them forever. For another, with the relative lack of good documentation in FOSS (and proprietary) projects, the existence of complete, clear, and maintained documentation is a strong differentiator. Finally, people talk about and share projects that empower them through documentation, making those projects literally remarkable. This word-of-mouth marketing is priceless.

Don't be among the host of companies that shortchange their FOSS projects by not providing adequate support. Hire dedicated tech writers and blow away the competition.

a. <https://www.intel.com/content/www/us/en/developer/articles/community/annual-intel-open-source-community-survey-results.html>

28. <https://producingoss.com>

29. <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>

30. <https://fossforge.com>

31. <https://thegooddocsproject.dev>

Prepare Messaging

Unless the team has decided to throw the project over the wall (see [Choose a Release Style, on page 320](#)), you'll undoubtedly want some sort of announcement or messaging strategy for when you make the project publicly available. Effective writing takes time and effort, so don't leave it for the last minute. Start thinking about and working on it as early as possible.

The messaging needed depends upon the strategic goals of releasing the project as well as the target audience. Don't treat each goals + audience combination the same as every other, copying what worked elsewhere. Verify that the messaging says what's necessary to get the attention of the target audience, using language they'll understand and appreciate in places where they're known to be looking.

This is a job for Marketing, who hopefully already has a representative on the release team. Work with them to develop the messaging and prepare it for the appropriate channels. Remember: while you may not know marketing very well, they may not be familiar with FOSS and its communities. You'll need to collaborate and learn from each other to make the messaging successful.

Decide Whether to Use a CLA/DCO

The “maintain control” and “open governance” release styles, require a decision about how to handle contributor copyright and licensing permissions. This decision isn't usually necessary for projects that are thrown over the wall, as there's no intention there to accept contributions. Nor is it needed for projects handed to foundations, as the receiving nonprofit will already have its own preference and/or policy with regard to contributor copyright management.

The three common options for this management are the following:

1. Have contributors sign a Contributor License Agreement (CLA).
2. Require Developer Certificate of Origin (DCO) signoff for all contributions.
3. Give it a miss.

As you've come to expect, each option comes with trade-offs.

Have Contributors Sign a CLA

You learned in [Contributor License Agreements, on page 34](#), that the primary goal of a Contributor License Agreement is to minimize any potential legal complications from distributing those copyrighted works. By signing a CLA, the project contributor affirms:

1. They have the right to share the IP that's in the contribution.
2. They give the project license to alter, distribute, and administer their contribution.
3. The contributor will never revoke that license.

You may read this and think, “Yes, that's exactly what we want! Let's do it!” but I caution you to consider the options before making a decision. CLAs, in particular, come burdened with some unique complications.

For one thing, a lot of administrative overhead is necessary to use CLAs in a project. The signed CLAs require storage, and because the CLAs include Personally Identifying Information (PII), that storage must be secure. While secure, there must be a way to verify signature status before accepting any contribution. Those contributions that arrive from people or organizations who don't have a signed CLA on file must be rejected. It's possible to automate this, but it's still a complex process.

For another thing, many people won't contribute to FOSS projects if doing so requires they first sign a CLA. They feel that CLAs create burdens and delays in the contribution process, and they're not necessarily wrong about that. Furthermore, if that CLA is for a corporate-sponsored or single-vendor FOSS project, potential contributors will walk away rather than risk donating their time and skills to a project that may use the CLA-granted permission to lock their contribution behind a proprietary license later.

Despite these complications, many corporate- and foundation-sponsored FOSS projects still require CLAs. They see these trade-offs as worth the extra effort.

Require DCO Signoff for All Contributions

The DCO is a more streamlined alternative to CLAs. If you recall from [Developer Certificate of Origin, on page 35](#), with a DCO the contributor confirms that they have the right to distribute their contribution and that they do so under the same conditions as the license of the project. In theory, this provides some IP insurance for the project receiving the con-

tribution. In practice, the picture isn't so clear. Requiring a DCO for contributions has at least five downsides:

1. *It's self-attested.* It's not that people necessarily *intend* to lie about having permission to share the IP in the contribution, it's just that they may sign the DCO without reading or understanding what it means. This calls the validity of the attestation into question.
2. *It's tricky to apply to contributions outside of a VCS.* The DCO requires an electronic signature using the project's Version Control System (VCS). Contributions outside of that system therefore can't be signed in the same way or perhaps will not be signed at all.
3. *It's not implemented in VCSs aside from Git.* The most popular VCS in the world right now is Git. It includes a way to provide the digital signature required for the DCO. Other VCSs don't appear to, though it may be possible to come up with a process for providing the signature via some other means.
4. *All contributions still need to be checked for a signature before acceptance.* Just as with CLAs, someone or something must review each contribution prior to acceptance to ensure the required signature exists. Failure to do this can cast suspicion on the entire contribution pool: does each have a signature or not?
5. *The DCO hasn't yet been tested in the courts.* The DCO is a legal document wherein someone attests they have the right to share intellectual property. As a legal document, eventually it will be tested in court to see whether saying "Yeah, I have permission to do this" is enough to protect the project receiving the contribution from accusations of copyright infringement. Experts appear to feel that the legal risk here is minimal, but it's not zero either.

Again, we have a situation where the entities controlling projects have found that these complications are safe enough for their particular risk profiles. In fact, because of its low administrative overhead compared with CLAs, the DCO is slowly becoming the preferred method for those entities that feel the need to manage contributor copyright and licensing permissions.

Why All the Copyright Fuss?

For decades, Free Software projects rolled along nicely without any sort of contributor copyright management. Then came the SCO-Linux court cases,^a wherein the SCO Group argued that it holds copyright and has trade secrets within the UNIX operating system, and those copyrights and secrets found their way into Linux. This shone a spotlight on the previously neglected matter of copyright ownership in FOSS contributions, leading to the creation first of CLAs and later to the DCO.

These cases are infamous in the FOSS ecosystem, partly due to how long this suite of cases has continued and partly due to the common perception that SCO is just flat-out wrong. Common perception, though, is not the law, and so the cases continue through the appropriate legal channels.

a. https://en.wikipedia.org/wiki/SCO-Linux_disputes

Give It a Miss

The final and most common option is to not manage contributor copyright and licensing permissions at all. Throw all caution to the wind and give it a miss. Depending upon the risk profile of your organization, this option may send chills down your spine, and yet it's how the majority of free and open source software projects have operated and still do operate, and quite smoothly at that.

When considering the matter of CLA versus DCO, most projects decide both are more trouble than they're worth. Each is uniquely high maintenance and neither provides enough (or perhaps any) value to the projects to implement them. They're found lacking in those projects' cost-benefit analyses.

Instead of managing the administrative overhead of a CLA or DCO, these projects work under the Inbound Equals Outbound assumption: the license under which the contributor provides their contribution (inbound) is the same as that of the project itself (outbound). For the most part, this assumption works surprisingly well. However, it's still an assumption, and undocumented assumptions are dangerous. You should always make your expectations clear, and inbound contribution licensing is no different. Convert that assumption into a fact; document any inbound = outbound requirement in the project's contribution guide.

Don't overlook this option, especially but not exclusively for low-risk projects that your organization releases. As you'll learn in [Relicensing Open Core, on page 383](#), these contributor copyright management tools tend only to be useful when relicensing a project. Since that is (or should be) a rare occurrence, you may likely be able to avoid the administrative headaches of CLAs and DCOs.

Track Metrics and Measure Value

Having worked through [Release Strategically, on page 301](#), you likely already have a firm grasp on the value that releasing this FOSS project should bring to your organization. How can you prove that? By measuring it, right from the very start of the project.

Few companies think about metrics until much later in the FOSS project life cycle, which is a mistake. The early days of any program or project—FOSS or otherwise—are when things are most likely to start going off course. If you're not already paying attention to data about the direction of the project, you'll never be able to chart course corrections.

What data can help the team determine whether the project is on track to providing the expected value? The answer to that question depends entirely upon what the company expects to get out of releasing the project. So many options exist here that it can be difficult even to see what they are. The CHAOSS Project has collected and documented dozens of different metrics,³² which helps immensely when you're starting this process. It even provides tools that help you gather and store the necessary data³³ so you won't need to create tooling from scratch.

It can be tempting, when faced with a buffet of options like the CHAOSS metrics, to try to load your plate up with all you can and start chasing down every possible data point. I encourage you to resist that urge and instead focus efforts at first on only the metrics most relevant to the expected value. Start small, especially while you're still new at this.

Choose Release Location

OK, so your organization is going to release a FOSS project. Good for you! Where are you going to release it to though? Where will people look to find this project? It's time to select the project's release location.

The release style you selected will matter here. If you've chosen to donate the project to a FOSS-related foundation, then it may have its own location for the project. However, you won't know this for sure until you've had the opportunity to choose and apply to a foundation. All other release styles though—and even some foundations—require choosing a home base for the project. We call this home base the *forge*.

32. <https://chaoss.community/kb-metrics-and-metrics-models/>

33. <https://chaoss.community/software/>

Selecting a Collaboration Forge

If you remember from [Code Forge?, on page 117](#), a forge is where people come together to access and collaborate on a free and open source project. There have been many forges over the years of FOSS. The most popular for-profit forges right now are GitHub³⁴ and GitLab.³⁵ You will also find a few nonprofit forges, such as Codeberg,³⁶ and it's even possible to self-host a forge using a FOSS project like Forgejo,³⁷ which is itself hosted at and is the software at the heart of the Codeberg forge.

All of these forges are based around the Git version control system³⁸ and offer additional features such as issue trackers, project management utilities, access control, and build pipelines. These features are all similar enough that it's relatively easy for people familiar with one forge to function well in another, meaning that the feature sets aren't much of a differentiating factor. In fact, as you compare the forges, you'll find few differences between them:

- *Reach or visibility*: Will the forge make it easier for people to locate the project? To some extent, this factor is a red herring. With modern search engines, it's relatively easy for people to locate what they need. Also, if the project has a website, it'll contain at least one link to the forge.
- *Ease of integration and administration*: How hard is it to integrate necessary tooling and manage administrative matters, such as access control. While all of the forges will handle these things slightly differently, they all have adequate capabilities—including API access for automation—for all but the most complex projects.
- *Costs*: All of these forges provide free access for FOSS projects.
- *Availability or reliability*: At the time of this writing, none of the listed forges are any less reliable than the others. Even a self-hosted Forgejo instance provides reliable access if properly secured and kept up-to-date.

So if all of the forges function more or less equally well, have similar minimal financial costs, and are stable and reliable, how are you to choose? Revisit your goals and constraints.

34. <https://github.com>

35. <https://gitlab.com>

36. <https://codeberg.org>

37. <http://forgejo.org>

38. <https://git-scm.com>

Consider Ecosystem Resiliency

GitHub is currently the most popular forge for FOSS projects, so many people will assume that's where the project lives. As the most popular, it's also the one about which people write the most articles, which may prove helpful as you're getting started. For many, they have enough already on their plate and their constraints prevent them from taking on too many more decisions. Others may already use GitHub Enterprise inside their organization, meaning they have access to people already familiar with administration of the tool. This can save a lot of time when releasing and maintaining a FOSS project.

As the most popular forge—which for some has become synonymous with “open source” despite being a proprietary service—many people and organizations simply default to GitHub when releasing projects. Functionally, this is a good choice, but strategically, another may be better. Having so large a proportion of FOSS projects hosted in a single forge creates a single point of failure (SPOF). Regardless of the ecosystem, SPOFs are problematic. For a healthy ecosystem, you never want to be in a situation where the entire ecosystem builds upon a single component. This lack of resiliency places the ecosystem at risk. We see the real potential of this a few times a year when the GitHub service becomes inaccessible and open source contributors all over the world must pause their work until the service returns to normal.

That GitHub is occasionally inaccessible isn't necessarily a problem or a black mark against this important online service. All services experience periodic outages, and, as you learned previously, the reliability of these forges isn't much of a differentiator between them. Resiliency, on the other hand, may be. Unless your organization's strategy and budget, time, and skill restraints dictate otherwise, I encourage you to make choices that increase rather than decrease the overall resiliency of the FOSS ecosystem. Don't default without thought; consider all the options.

Selecting a Foundation

If the work you did in [Choose a Release Style, on page 320](#), led you to decide to donate the project to a foundation, you're now faced with another decision: which foundation will you choose as the home for the project?

You've already learned in [Donate to a FOSS Foundation, on page 311](#), that many different FOSS nonprofit foundations exist in the world, and new ones launch every year. You can find lists of nonprofit organizations in several resources, including these:

- The FLOSS Foundations Directory³⁹
- The Europa International Foundation Directory⁴⁰
- The Wango Worldwide NGO Directory⁴¹

Of these, only the first exclusively lists foundations dedicated to free and open endeavors, so it may be the most useful in this context. However, as none of these lists are identical you may still find yourself searching the other resources, depending upon your foundation criteria and constraints.

Some factors you may wish to research when evaluating foundations include the following:

- *Area of interest:* Some foundations specialize in supporting FOSS projects related to specific industries or themes. For instance, the Apereo Foundation⁴² supports FOSS related to higher education and academia; NumFocus⁴³ is dedicated to projects related to scientific data, research, and computing; OW2⁴⁴ provides a home to middleware and business applications. Other foundations—including the Apache Software Foundation,⁴⁵ the Eclipse Foundation,⁴⁶ and Software in the Public Interest⁴⁷—provide support to many different types of projects.
- *Transparency:* All FOSS foundations are not-for-profit entities. Depending upon where a foundation is incorporated, it may be required to make its finances public via government filings. For better or worse, most of these foundations are incorporated in the United States. For these, you can find the most recent public filings on Guidestar⁴⁸ or the Nonprofit Explorer from ProPublica.⁴⁹

39. <https://flossfoundations.org/foundation-directory/>

40. <https://www.routledge.com/The-Europa-International-Foundation-Directory-2022/Publications/p/book/9781032274300>

41. <https://www.wango.org/resources.aspx?section=ngodir>

42. <https://www.apereo.org>

43. <https://numfocus.org>

44. <https://www.ow2.org>

45. <http://apache.org>

46. <http://eclipse.org>

47. <https://www.spi-inc.org>

48. <https://www.guidestar.org>

49. <https://projects.propublica.org/nonprofits/>

- *Region of operation or incorporation:* For some, it's important to keep data and other operations within the same region in which they're based. The digital sovereignty movement in the European Union⁵⁰ is an example of such a concern in action. Some companies may wish to keep this consideration in mind for political, regulatory, or convenience reasons.

All foundations will have their own criteria for accepting a project (and some may not accept projects at all), one of which typically is financial support of some sort. If you're going to ask them to take care of your puppy, it's fair of them to ask you to help foot the bill. Financial support often takes the form of corporate membership in the foundation, which usually brings additional benefits, such as logo placement on the website or discounted conference registration fees. The cost of membership can be fixed or depend upon a variable such as headcount or revenue of the member company. Even if financial support isn't a requirement for a foundation accepting a project, it's still a best practice to provide it, if only to ensure the longevity and sustainability of the foundation and therefore also the project.

As you and the team are evaluating foundations, maintain a ranked list of options rather than choosing only one candidate. The relationship you're looking to establish with a foundation is two-way, and your company's interest must be reciprocated by the foundation. A foundation may have many reasons to opt not to accept the project you wish to donate to them, including a lack of resources to provide adequate support to the project or the project not being a good fit for the foundation and its mission. So it's always a good idea to have backup candidates, just in case.

Select a License

Thanks to decades of fixation on licensing, "Let's pick a license!" is usually the first thing people think when they decide to release a FOSS project. In reality it's one of the final steps in the entire releasing process. That's because choosing the correct license depends on everything that we've covered up to this point in this chapter as well as in [Chapter 19, The Strategy of Releasing FOSS, on page 297](#). Now it's time to pull all of these previous decisions together. Doing so usually makes the choice of the correct license for your company's business needs obvious.

As you're pulling these decisions together, do so in collaboration with your legal team. Remember, licenses are legal documents that help to manage

50. [https://www.europarl.europa.eu/RegData/etudes/BRIE/2020/651992/EPRS_BRI\(2020\)651992_EN.pdf](https://www.europarl.europa.eu/RegData/etudes/BRIE/2020/651992/EPRS_BRI(2020)651992_EN.pdf)

some of the complexities of copyright and intellectual property. Individuals can get away with releasing FOSS projects without legal advice, but a company should never take that path.

As you probably remember from [Types of FOSS Licenses, on page 44](#), FOSS licenses are of two types. Oversimplified, these are *reciprocal licenses* (share and share alike) and *permissive* (do what you like but give credit where credit is due). Four primary factors influence not only which type you choose but also which specific license from that type:

- Project components
- Foundation requirements
- Business needs
- Organizational values

Components

The licenses of the components used in the project you're releasing can be a major factor in which license you'll apply to the final released project. Using the information you learned in [Chapter 11, Know the Links in Your Software Supply Chain, on page 173](#), you can gather a list of components used in the project. Now review the license for each one of them. Do any of those components use reciprocal licenses? If so, the reciprocity requires that derivative projects using these components must also be licensed reciprocally.

The key word there is *derivative*. Remember from [Chapter 3, Licenses: The Rules of IP Engagement, on page 39](#), a derivative in FOSS is a modification to an existing copyrighted work or a work that incorporates some or all of another copyrighted work. Whether a project is a derivative of its components depends upon how the component is used or integrated into the project. Revisit [Derivatives, on page 58](#), to learn about the complications in defining what is or isn't a derivative work.

Regardless of whether a project is a derivative of its components, it must still respect the terms of the licenses of those components. Some of those licenses may not be compatible with other licenses, which may impact how you license the project. For instance, if a component is under the Apache-2.0 license, compatibility issues will prevent you from releasing your project under the GPL-v2.0 license. The [FOSS License Compatibility Matrix, on page 420](#), can help you work through these licensing questions.

Good Citizens Include an SBOM

These components are going to be a lot easier to review if you received a list of them via an SBOM, which you learned about in [Software Bill of Materials \(SBOM\), on page 194](#).

What's that? None of the dependencies for your project came with an SBOM, so you'll have to build one from scratch? That's not too surprising. SBOMs are only at the start of their broad adoption in the free and open source software world, but that adoption is growing rapidly every year. Therefore, it's a good idea to be a part of this positive change and include an SBOM in any FOSS software project that your organization releases.

It's possible that the tools your company uses to manage its software supply chain (see [Tools, on page 192](#)) will include SBOM creation functionality. If you use the REUSE tools for applying the copyright and license statements to the source code (you'll learn about this in [Software Source Code, on page 351](#)), then you also can use its SBOM creation options.^a

a. <https://reuse.software/faq/#bill-of-materials>

Foundation

Many foundations have intellectual property policies that include a preference for license in the projects they host. For instance, Software Freedom Conservancy prefers to support projects that are reciprocally licensed,⁵¹ while the Apache and Eclipse foundations prefer their own eponymous licenses.

If your company has chosen to donate a project to a foundation, and the foundation has accepted the project, follow the foundation's guidelines for licensing rather than licensing first then having to change it for the foundation later.

Business Needs

As you'll learn in [Chapter 22, Incorporate FOSS into Business Models, on page 369](#), some companies release FOSS projects as key elements of their business model. For these companies, a certain license may be required to support that model.

For instance, companies that rely upon a patent portfolio for a portion of their revenue may require the components and projects they use and release be available under licenses with explicit patent clauses, such as

51. <https://sfconservancy.org2>

Apache-2.0. An open core company that's built a freemium SaaS business model based around a FOSS project they created may wish to release that project under the Affero General Public License (AGPL) to ensure that any derivative (and perhaps competing) projects also and reciprocally be released under AGPL.

Organizational Values

The value of the organization releasing the project can impact choice of license. Organizations that value the consistent availability of resources, such that all are free to benefit or profit equally and in perpetuity, tend to prefer reciprocal licenses. Mission-driven organizations, such as B Corps or NPO/NGOs, often release their projects under these licenses, as this ensures their work will remain freely available for as long as derivatives continue being made from it.

Permissive licenses with explicit patent grants tend to be preferred by for-profit entities. These permissive licenses—Apache-2.0 in particular—often are called “business friendly” because of this patent grant, leading many companies to disregard any other licenses.

This is a shame, as the “business friendly” label is somewhat deceptive and certainly unnecessarily judgmental, implying that other FOSS licenses are not suitable for business use. Success in business hinges upon the business model and related operations. The license of a work that's provided for free is a relatively small factor in any well-run business. Nothing's inherent in a FOSS license, such as the GPLv3—which includes an explicit patent clause but somehow is never included in that list of “business friendly” licenses—that would make a business less successful for using it rather than Apache-2.0.

Should the organization foresee a future where they'll need to relicense a project, “closing” it under a proprietary license for business reasons, then it should release it under a permissive license that will make that relicensing easier, such as MIT. However, if the company already foresees a possible need to relicense, it should question the business value of releasing the project as FOSS at all.

The More the Merrier?

You learned about multi-licensing back in [Too Much of a Good Thing?](#), on page 57, but to recap: a copyright holder may release a work under two (or more!) licenses if they want. When they do this, the receivers of the work may choose at any point under which license they're using the work. This is an option you can pursue with your own projects.

Some companies follow this path with their single-vendor FOSS. They release the project under a strong or network reciprocal license (see [Reciprocal \(aka Copyleft\) Licenses](#), on page 44), allowing anyone to use and modify the project but ensuring that any derivatives must provide the same terms as the original license. At the same time, some may choose instead to pay a fee to access the project under a proprietary license. The terms of this closed license may include the ability to release derivatives under nonreciprocal—and most likely proprietary—licenses.

This multi-licensing approach is useful in a *freemium* business model, which you'll learn more about in [Chapter 22, Incorporate FOSS into Business Models](#), on page 369. Oracle uses it to good effect with its MySQL database. While the project is available under the reciprocal GPLv2 license, Oracle also offers several other subscription licensing models to support their customers' varying business needs.

Things can get complicated when making a project available under multiple licenses, so make sure you consult with a specialist as well as legal counsel before doing so.

Apply the License

Once you've selected a license, it's time to apply it to the project. Many people and organizations get this step wrong, despite it being relatively easy (if perhaps tedious at times).

It's important that whoever uses the project, however they acquire it, always has access to the license under which they have permission for use. This not only provides a way for them to review the terms they've agreed to by using the project, it also provides proof that they have that permission at all. This is especially important in highly regulated industries, where audits are common, risks are high, and everyone needs to make sure they can verify the provenance and license of all components.

Therefore, properly applying the license to the project protects not only the intellectual property rights of those who release it but also the rights of those who use or acquire the software. Applying the license takes three steps:

1. Add the license to the repository/forge.
2. Point to the license in the README.
3. Add Copyright and License Statement to the work itself.

Let's go into each of these in more detail.

Add License to Repository/Forge

A copy of the exact and complete text of the license must go into the repository or forge where the project will reside. Typically this text is in a file named LICENSE, LICENSE.txt, or LICENSE.md, but for historical reasons, in the case of the family of reciprocal GNU General Public Licenses (GPL) the file is named COPYING.

It can sometimes be more difficult than you'd expect to locate the canonical text for the license you've selected. I recommend the [SPDX License List](https://spdx.org/licenses/)⁵² as a resource for this. Click the license entry in the list, then navigate to the canonical link(s) listed under "Other web pages for this license."

When doing this, you'll likely notice that the text of the license is *right there* on the SPDX License List page, and you'll be tempted to cut/paste it into your LICENSE file. You don't want to do this, trust me. This text often contains special encoding that the SPDX team uses for license matching. Also—while unlikely—there's always the chance that the canonical text of the license has diverged somehow from the text that SPDX has on file. Should you find an instance of such divergence, be a good FOSS citizen and let the SPDX team know by opening an issue in their GitHub repository.⁵³

Point to the License in the README

Now that you have a LICENSE file that will travel along with the project, make it easier for people to locate.

In FOSS projects, it's best practice to have a file named README or README.md that acts essentially as a greeter, welcoming people to the project and providing vital information. Part of this vital information is the license providing the terms and permissions for use of the project.

Somewhere in that README file, mention what the license is for the project and where people can find the text of that license. It doesn't need to be anything fancy or complicated. For instance, here's an example in Markdown format:

```
# License and Copyright
```

```
All content is copyright the contributors and is available under the
Creative Commons Attribution-ShareAlike 4.0 International license.
```

```
Please see the [LICENSE](./LICENSE) file for complete information.
```

52. <https://spdx.org/licenses/>

53. <http://github.com/spdx/license-list-XML/>

Add Copyright and License Statement to the Work

Pointing to the LICENSE file in the README is a good step, but it's not enough. You also need to add a copyright and license statement to each and every file in the project.

Yes. Each and every file.

The reason for this is that no rule says that people must take either all or nothing of a project. If they feel like it, they may cherry-pick the files that suit their purposes. Not having the copyright and license statement in every file therefore leads to all-too-common conversations like this:

IRENE, Legal: Hey, Andrea, where did this template come from?

ANDREA, Programmer: Oh, I copied it from some open source project somewhere. Pretty cool, isn't it?

IRENE: Yes, yes, very nice. Do you know which project it's from?

ANDREA: Nah, it's just something someone mentioned on StackOverflow or somewhere like that.

Irene now faces a file of unknown provenance with an equally mysterious license—or perhaps no license at all. Either way, it spells immense headaches for anyone who's trying to manage compliance or software supply chains.

The statement added to each file should contain two pieces of information: who has copyright on the file and the license under which it's released. How you format and convey that information will vary by the contents of the file.

Copyright All of Us

When you first release the project, all of the copyright probably belongs to the company. Therefore the copyright portion of the statement could be something like "Copyright Bees4Less <ospo-licensing@bees4less.com>."

As the project evolves, others will contribute their own copyrighted material to the files. Multiple copyrights in a single file. Mass hysteria!

It's OK. FOSS has this covered. Thanks to version control systems, it's easy to see who's made changes to (authored) a file. Therefore, the copyright statement can say, "Copyright the Authors" or even "Copyright Bees4Less <ospo-licensing@bees4less.com> and the Authors," and people who need more information than that are able to locate it.

Feel free to start out by using the company name in the copyright statement if you'd like—it's good branding, after all—just remember to change the statement once a file acquires more authors.

Software Source Code

The most common case for adding copyright and license statements for FOSS is for software source code. It's also therefore the case that's best defined. For that, we can thank the fine folks at Free Software Foundation Europe (FSFE) and their helpful REUSE project.⁵⁴

REUSE provides a standardized format for copyright and license statements along with a collection of tools to help you create and add those statements to the software source files in a project. The standardized format, as well as the use of SPDX license identifiers,⁵⁵ makes life considerably easier for creating SBOMs or performing other software supply chain operations. Revisit [Software Bill of Materials \(SBOM\)](#), on page 194, for more information about this.

The basic format is simple, using two tags: `SPDX-FileCopyrightText` and `SPDX-License-Identifier`. Here's an example:

```
/*
 * SPDX-FileCopyrightText: Bees4Less <ospo-licensing@bees4less.com>
 * SPDX-License-Identifier: GPL-3.0-or-later
 */
```

The statement should be added to the top of every source code file, commented out. The example here uses C-style multiline commenting, `/* ... */`, but the comment style will depend upon the programming language of the source code.

Please note that the stewards of some licenses—notably the GPLs⁵⁶ and Apache⁵⁷—have provided their own instructions for how to apply their licenses. Ideally, we all should seek out and apply these instructions for each license used in our projects. Realistically, that doesn't happen. Practically, it *shouldn't* happen. The ecosystem-wide benefit of having a standardized method for applying copyright and license statements, thereby easing the software licensing compliance and supply chain burden, outweighs the need to seek out and customize these statements for each license in the project. Use the REUSE format.

Creative Works

By its very nature, creative works are harder to add copyright/license statements to than source code. While code is all text-based, many creative

54. <https://reuse.software>

55. <https://spdx.dev/learn/handling-license-info/>

56. <https://www.gnu.org/licenses/gpl-howto.en.html>

57. <https://www.apache.org/legal/apply-license.html>

works are binary. This means you can't simply add a commented-out block of text to the work; you need to find other ways to include this information in each file.

The best way to do that is by including it in the file's metadata. This is a type of data that provides information about the contents of the file. Many file formats include metadata fields. Some of these fields contain autogenerated information like bit rate or creation date and time, but other fields can contain custom data. Sometimes the format will even include a dedicated field for copyright information. Detailing the metadata options available in various binary formats is thankfully outside the scope of this work, but you can find plenty of information online about the format with which you're working.

Document formats like Word also often include fields for metadata, though depending upon the format and application used, it can be tricky sometimes to find and modify them. I encourage you to try if you have the opportunity but also to include the copyright and licensing information in a more accessible form, such as in the document footer, in the frontmatter, or—for text-based formats like HTML or Markdown—in commented text blocks.

However you choose to include this information in creative works, verify that method with your legal team. They'll verify whether your planned method covers all the appropriate legal bases.

The Copyright Dating Scene

Question: Don't I need to include a date in the copyright statement?

Answer: Nope.

Nearly all jurisdictions use "author life plus X years" as the duration for copyright protections. For example, right now in the United States an author receives copyright protection for their work for the duration of their lifetime plus another 70 years.

So the date a work is created doesn't matter. The date the creator dies does, since that's when the "plus 70 years" clock starts counting down. If someone does end up needing a creation date for any reason, most of the time they can find it in the version control system for the project.

So go ahead and leave that copyright year off your copyright and licensing statements. It's OK to do so, and it also removes the burden of wondering when (or whether) you should update it.

Comply with Component Licenses

In [Components, on page 345](#), you reviewed the licenses of all project components to ensure the license you select isn't in conflict. Now it's time to comply with the licenses of those same components.

The most common responsibility required by FOSS licenses is *attribution*. You learned a bit about this already in [Give Attribution, on page 206](#), which also contains a couple of examples of attribution listings. Each license will tell you what sort of information you need to provide with the attribution, but usually it's the name and copyright/license statement of each component.

Most often, software projects collect and publish these attributions in a file in the repository on their forge. This file can go by several names, but Apache popularized the use of NOTICE,⁵⁸ and it's one of the most commonly used names. I recommend sticking with that name rather than coming up with your own.

If your organization uses tooling—like those listed in [Tools, on page 192](#)—to keep track of its software supply chain, those same tools may be able to generate a NOTICE file for you. Look for this feature before doing a lot of manual work cutting and pasting copyright and license information into a text file.

Prerelease Checks

Every pilot does a preflight checklist before taxiing down that runway. You should similarly do a little safety check before releasing a project to the public.

Take a quick tour through the project, and verify someone has had a look at and signed off on each of these topics:

- Security
- Documentation
- License and its application
- Patents
- Trademarks

The odds are good that after all of the work the team has put in, everything is in order. Still, it never hurts to check.

Verify P&P

Take another look at any policies and procedures (P&P) that touch upon this effort.

58. <https://infra.apache.org/licensing-howto.html>

You and the team have done a *lot* of work getting this project ready, and some of that work may have involved changing or creating how and why things are done. If this new and improved information doesn't match what's contained in existing P&P documentation, a whole lot of people will understandably be frustrated down the line.

Review the existing P&P and verify it's up-to-date and contains all of the information required to set people up for success once the project goes public. Fixing these things now is going to be a lot easier than doing it later, after you've had to answer the same question several times due to incorrect documentation.

Verify Messaging Is Ready

The work the team did to [Prepare Messaging, on page 336](#), will pay off now. Because you've been working on it this whole time, the messaging will likely be ready to go. Make sure everything is still on track and coordinate with Marketing on release and announcement timing.

Prepare the Forge

If you haven't done so yet, ensure that the forge is ready for collaborators to join. This process includes fundamental forge administration, such as setting up access and authorization, and any related team definition required to make contributing easier.

It's also helpful at this time to place the project on the forge but *keep it private*. This allows you and the team to stage the project for release while also doing any last-minute tests that may occur to you.

Once all this is done and you're confident (enough) things are ready to go, it's time.

Release!

Make the project public!

On most forges, this is a simple option that you toggle to enable anyone to view the project, not only those who have special access. It can feel somewhat anticlimactic in that way, since minimal fuss or drama is involved, but that just means you and the team have done a great job preparing the project for release.

Congratulations! Your organization has released a FOSS project! If you're throwing the project over the wall, your work is now more or less done. Any of the other release styles will, like a puppy, require on-going care and feeding. You'll learn about that in the next chapter.

Engage with Your FOSS Projects

Unless your organization has chosen to throw the FOSS project over the wall, releasing it is only the beginning of its life cycle. Like that metaphorical puppy, the project requires attention to keep it healthy and happy. This chapter introduces you to some facets of FOSS project maintenance and management.

Many resources about how to maintain and operate FOSS projects are already available, so you'll learn the basics here to get you started, and you can seek out these other resources—found in [Appendix 5, Other Useful Resources, on page 423](#)—for in-depth coverage. In particular, have a look at *Program Management for Open Source Projects: How to Guide Your Community-Driven, Open Source Project* by Ben Cotton,¹ which presents FOSS maintenance in an approachable and pragmatic way that's accessible to people of all levels of experience.

Get the Word Out

To paraphrase *Field of Dreams*,² “If you build it, they will *not* come.” Releasing a project usually isn't enough to attract people to use and contribute to it, since they're unlikely to realize the project even exists. To raise that sort of awareness, you need to get the word out about the new FOSS project. You need marketing.

Build on the Existing Foundations

The work performed to [Prepare Messaging, on page 336](#), is a good foundation on which to build ongoing marketing efforts. Collaborate with Marketing to continue that work after your organization has released the project to the public.

-
1. <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>
 2. https://en.wikipedia.org/wiki/Field_of_Dreams

If you're fortunate, your company will not only have an Open Source Program Office (OSPO) (or its equivalent under a different name) but also at least one person filling the role of [Technical Marketer, on page 115](#). This person bridges the marketing skill set and an understanding of FOSS and its communities, and they're worth their weight in gold when raising awareness of a new free and open source project. They'll already be familiar with the channels where potential contributors will see the messaging as well as how to speak to these people to gain their respect and attention.

Don't worry if you don't have an OSPO or tech marketer to lend a hand; the marketing team in the organization know their business and will be invaluable. However, they may not yet be acquainted with FOSS communities and will need help getting up to speed. Collaborate with them to help them understand this new-to-them market and craft authentic messaging.

Use Common Channels

Certain marketing channels are common in the FOSS world and therefore likely to be effective for getting the message out about the project.

Project Website and Blog

If a project is more than a small utility or library, it's helpful for it to have its own website and associated blog. These can receive search engine optimization (SEO), hopefully making it easier for people to locate the project with their favorite search engine. Marketing will know all about how to do this.

The website acts as the “storefront” of the project, welcoming people and giving them vital information, such as the following:

- What the project does and the value it provides
- Where to find the project's forge
- Where to find documentation and community resources

The blog is helpful for sharing regular updates with community members and people using the project. These updates can include release announcements, tutorials, updates about ongoing development, or any number of other topics of interest to those engaged with the project.

In the case of projects released under the Release but Retain Control release method (see [Release but Retain Control, on page 309](#)), the project website and blog may share namespace with your organization's domain name (for example, bees4less.com), but more often the project will have a

separate domain name (openhive.foundation). This can help in at least three different ways:

- It aids in discoverability and provides a stable home on the internet for the project.
- It signals to the public that the project, while supported by the founding organization, operates separately as a community.
- It can minimize community and project disruption should your company donate the project to a foundation in the future.

If the project website and blog have a separate domain name, make sure it gets into the standard domain registration renewal process with IT. All too often someone outside of IT registers and maintains the project domain, then when they leave the firm (or simply when they're not paying attention) the domain name expires, causing no end of complications. Work with IT to ensure this never happens to your project.

Article Placement

Some projects connect with potential community members by working with journalists and publishers to place articles in online or print publications. This can be an effective way to introduce the project to a lot of people who might otherwise not know to look for it.

While your company may end up paying for article placement, it's important that articles not read like sales pitches. This can be difficult to do, since often the people writing those articles are more familiar with writing sales copy than purely informative works. To help with this, and to add authenticity to the piece, try to have a community member who's external to your organization tell their story of the value the project brings to them. Real-life stories of how a project solves someone's problems are interesting and compelling.

If possible, negotiate with the publication so you can republish the article on the project's blog after a period of time. Regardless, the community should share links to the article with their networks to help spread the word and drive more readers to the publication, raising the profile of the project in the process.

Events

Events—conferences, hackathons, meetups, and the like—are a time-honored approach to promoting free and open source projects. Thanks to its community-centric nature, people have gathered to share their passion about FOSS for nearly as long as it has existed.

These gatherings don't need to be dedicated to a single FOSS project (or dedicated to FOSS at all). Even small events like meetups require a lot of time, talent, and treasure to organize and run, so often an event will be more general in nature, welcoming participants with a shared interest in a higher-level topic. Thus you'll more often find, for instance, ongoing Python meetups where people discuss the programming language along with components and utilities written in it than you'll find ongoing meetups dedicated only to the requests Python component.

Attending, presenting at, sponsoring, or otherwise participating in these events provides an audience you already know are predisposed to be interested in learning more about your project. If your company is fortunate enough to have hired a [Developer Relations/Advocate/Evangelist, on page 117](#), for its OSPO or another team, they'll be the best person to approach to come up with a strategy around promoting the project at events.

Paid Internship Programs

Paid internships are an underappreciated method of promoting FOSS projects. Some communities apply to receive interns from these programs so they can get additional assistance resolving bugs or feature requests, and the programs certainly can help with that. Less appreciated is the fact that by participating as a mentoring project in these programs, they're also broadcasting that they're investing in training the contributors of the future and, therefore, also in the sustainability of the project.

Several paid internship programs are currently operating in FOSS, including these:

- Outreachy³
- Google Summer of Code⁴
- Google Season of Docs⁵
- Linux Foundation Mentorship Programs⁶

Tens of thousands of potential interns apply to these programs, hoping to earn a spot to be paid to learn about and join a specific FOSS project and community. That's a lot of people who are learning about the existence of and value provided by your project. This makes paid internships a win-win situation: the project receives promotion and new contributors, and the interns receive training and a supportive new community.

3. <https://www.outreachy.org>

4. <https://summerofcode.withgoogle.com>

5. <https://developers.google.com/season-of-docs>

6. <https://www.linuxfoundation.org/about/mentorship-programs/>

All paid FOSS internship programs have their own timelines, criteria, and payment amounts. They also all require a fair bit of time from project community members to mentor and guide the interns. Before diving into the deep end of paid internships, take the time to plan and ensure the project can provide adequate support to the interns who'll be assigned to the project.

Choose Realistic and Meaningful Marketing Metrics

Depending upon the overall goals for releasing the project (see [Release Strategically, on page 301](#)), raising awareness of the project may be only the first of many tasks for Marketing, but it'll be an ongoing one. Collecting and watching the right metrics from the start will help ensure those goals are on track. Choosing which metrics are right, however, is easier said than done.

For one thing, you may find that people are unaware of or otherwise not in agreement on what those goals actually are. Pausing to meet and align everyone so they're aiming at the same targets can save a lot of frustration and arguments later on down the road.

For another, you may uncover a skills constraint. Marketing teams who are less familiar with FOSS will gravitate toward metrics that sound reasonable but in actuality mean little. These include the following:

- *Number of people using the project:* Unless the project includes some form of telemetry, this number is nearly impossible to measure in any meaningful way. If the project *does* include telemetry, for privacy reasons it's best practice to default that use-tracking to off. Therefore the data it collects still don't reflect anywhere near the number of people using the project.
- *Number of downloads:* While it's possible that this metric can act as a proxy for project awareness, in reality the data it provides may not be close to accurate. It's common for an individual to download a single project multiple times, and a person can download it from a different location than the one you're tracking.
- *Vanity metrics such as GitHub Stars:* The popular GitHub forge provides the ability for people to "star" a project as a way for people to bookmark projects so they don't lose track of them.⁷ Because the number of stars a project has received is public information, people often interpret the stars as a gauge of project popularity. This has led to illicit services that will sell or otherwise provide stars to projects, artificially inflating their perceived popularity.

7. <https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars>

These metrics are popular because they're relatively easy to collect, but as you see in the GitHub Stars example, each of them is just as easy to game. Their popularity and relative ease of automating also make these metrics more likely to be manipulated. Thankfully, we have metrics that can more authentically reflect whether efforts to raise awareness of the project are working. Here are a few:

- New people opening issues for the project
- New people contributing to the project
- Increase in diversity of people and organizations contributing to the project

As always where FOSS metrics are concerned, I recommend you visit the Community Health Analytics in Open Source Software (CHAOSS) Project.⁸ Its tools and metrics models are invaluable for anyone who's interested in metrics related to any sort of community, especially those of FOSS projects.

Manage the Community

Over the decades, free and open source software developed a reputation for toxic communities, and the sad fact is that this reputation was well earned. Thanks to negativity bias,⁹ it took only a relatively few bad actors, regardless of their minority, to give the entirety of FOSS this poor reputation. The rudeness of these people—ignoring questions or replying contemptuously, or acting in a racist, sexist, or otherwise bigoted manner—often was accepted as the price one has to pay for the freedom and openness of the software.

This has changed. While these bad actors still exist, their behavior is considered unacceptable by more and more communities. The adoption, enforcement, and acceptance of codes of conduct has helped immensely here by codifying the expectations for behavior within each community. For the most part, FOSS has become a welcoming and inclusive ecosystem.

That isn't to say that the work is done. Community management is now and always will be an ongoing endeavor rather than a one-time effort. Managing a community isn't only about preventing negative experiences or unwelcome behavior. It's also about ensuring people have what they need to meet their personal goals in the project. This may include things like mentoring, transparency, and leadership opportunities.

8. <https://chaoss.community>

9. https://en.wikipedia.org/wiki/Negativity_bias

Because it involves those squishy, difficult things called people, community manager is a complex and challenging role. It's also often a full-time one, especially as projects grow larger and more popular. Your company's OSPO may already have a [Community Manager, on page 116](#), on staff, but if not, then consider hiring one. Ideally, this person should be from within the project itself, since they'll already have built and earned the trust of the community.

Manage the Project

As with a community, a healthy FOSS project doesn't "just happen" without attention and intention; without this, chaos reigns. Ben Cotton summarizes it well in *Program Management for Open Source Projects*:¹⁰

Open source projects produce software, but they're run by people. People are dynamic and sometimes unpredictable. The more people are involved, the more unpredictable the project becomes. As a program manager, you bring order to the chaos. You don't do this by imposing order from the top down—that's not how open source projects work. Instead, you *give structure to the agreements of the community*.

The structure necessary varies from project to project, but all of it requires someone to pay attention and guide that structure so it fits in well with the overall project and community. An element of structure that's forced onto a project can ruin the whole, much like trying to force a mis-threaded screw into place.

The following are some of the structural elements that are common in FOSS projects of all sorts:

- Roadmap and release planning
- Bug triage and issue handling
- Translations
- Communication routes

That's an awful lot to consider, which is why it took an entire book to cover the topic. Rather than repeating everything that Cotton says in his work, I encourage you to read it to learn how to establish and maintain the appropriate structure for your project.

10. <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>

Stick a Fork in It

You learned in [Do You Have the Spoons to Deal with a Fork?](#), on page 45, that *forking* a FOSS project means to take a copy of the project, rename it, and start a new project and community around that copy. This is a natural part of free and open source and is allowed by all OSI-approved licenses.

A project fork happens for several reasons:

- Making a private fork of a project that's a component of a company product (though, as you may recall from [Upstream-First Policy](#), on page 272, this usually isn't a best practice).
- Forking a project to relicense it under a proprietary license.
- A community forking a project due to displeasure at how the project and/or community is run.

While all forks are somewhat complicated and messy, that last sort is especially so. As your company manages its project and its associated community, it should strive to avoid pissing people off. If it doesn't, those same people may fork that project and continue working on it without you.

This has happened several times in the past few years, largely due to formerly open core companies reassessing their business models and relicensing their software to be proprietary rather than open source (you'll learn more about this in [Relicensing Open Core](#), on page 383). Here are some notable forks in this category:

- OpenSearch,^a forked from Elastic's Elasticsearch^b
- OpenTofu,^c forked from Hashicorp's Terraform^d
- Valkey,^e forked from Redis's eponymous project^f

Forks aren't only something to worry about though. They're also a tool that your company can use to ensure the sustainability of its FOSS supply chain. Should an analysis of that chain discover a critical link that's neglected and in need of a new steward, your company can fork it and create a new, well-maintained project from it. Should you choose to do this, try to gain community support prior to forking. This transparency builds trust and can help attract contributors who wish to give their time and talents to a vibrant, active project and community.

-
- a. <https://opensearch.org>
 - b. <https://www.elastic.co/enterprise-search>
 - c. <https://opentofu.org>
 - d. <https://www.terraform.io>
 - e. <https://valkey.io>
 - f. <https://redis.io>

Enable Project Evolution

It can be tempting to think otherwise, but no useful project is ever perfect or truly “finished.” Ongoing project and community maintenance is a fact of FOSS life. While the books by Cotton and Fogel¹¹ will guide you through the details of this maintenance, it’s worth spending a little time now to consider a couple of common mistakes that companies make. These mistakes are two sides of the same coin: on one side the company isn’t putting in enough effort, and the flip side has them putting in too much.

Not Enough Effort...or Attention

A common misconception is that once a company has released a project, it no longer needs to pay much attention to it. “Our work is done; the community will take it from here.” This may be the case if you’ve chosen to throw the project over the wall, but all other release styles require ongoing oversight and hands-on contributions from the originating company.

It’s common for companies to start participating in their own project but then for that participation to dwindle as contributing staff are reassigned to other duties. This neglect can lead to project stagnation and eventually even abandonment of the project by contributors. The company has squandered its opportunity to grow a vibrant and strategically valuable community, and once that opportunity is gone, it’s unlikely to come again.

This dwindling attention can also lead to burnout of staff and community members. As company contributors are reassigned away from the project, those who remain often work harder in an attempt to hold the project and community together. This is exhausting and thankless work. Usually, it’s also futile. Once a company deprioritizes its participation in a project, it can be difficult to change the downward trajectory.

As you’ll learn in [Leave the Project, on page 364](#), it’s perfectly OK for a company to withdraw from a FOSS project, even one it released. However, turning your back on a project and allowing it to die from neglect isn’t the correct way to handle the situation.

Too Much Effort...and Control

Alternatively, some companies release projects and keep them in a command-and-control death grip. Instead of, as Cotton says, “giving structure to the agreements of the community,” in these projects the community has little to

11. <http://producingoss.com>

no voice at all. The company micromanages all elements of the project, from roadmap to release, afraid to collaborate with outside participants.

This lack of trust in outsiders is reflected and amplified back toward the company as potential community members grow to distrust it, its intentions, and its projects. Usually, companies are unaware that this distrust amplification is happening, and therefore they express frustration when releasing the project doesn't produce the expected strategic results. Not seeing a return on their investment, leadership withdraws support and resources for the project. Unless inspired and dedicated community members step in to resurrect it, the project is effectively dead.

Just Enough Effort

For a project and a community to grow and evolve, it needs the space to do so. It may take a little trial and error to figure out the correct balance for your company's project though.

Start by trusting in the community and sharing project responsibility with a few engaged and helpful contributors. It doesn't take many of these external contributors accepting a leadership ("core") position in the project to make a huge difference in how others perceive the company and its project. Actively seeking out collaborators and sharing responsibility signals to everyone that the project truly is open and wishes to accept their feedback and contributions.

These core contributors also provide continuity and sustainability for the project and its community. Even if the company deprioritizes its engagement, sharing responsibility with others outside of the organization can help ensure the project lives on should the company withdraw. This can help lengthen the company's legacy and impact, and it shares the project maintenance and the burden carried by internal company staff. This can reduce bottlenecks in the project and speed its development and evolution.

Leave the Project

It's OK for your company to wind down a project it has released, withdrawing from the community and investing no more resources in it. Companies don't like considering this option, not wanting to "waste" everything they've already invested. This behavior, also known as the Sunk Cost Fallacy,¹² leads a company to keep a project on life support, stumbling along beyond its natural days like some sort of FOSS zombie. Or if the company does decide to with-

12. https://en.wikipedia.org/wiki/Sunk_cost#Fallacy_effect

draw, they'll do so abruptly and cause damage to the project, its community, and the company's own reputation and brand.

Don't create a zombie project. Don't damage your brand. Take the appropriate steps, and the project will either fade away peacefully or be reborn under different leadership.

Be a Quitter

As counterintuitive as it may sound, one of the most common reasons for project failure is *not* quitting.^a Blindly following a path isn't dedication, it's foolish.

The most successful projects include milestones and checkpoints where the team evaluates whether they're on track and, if not, whether it's worth the effort to continue. This is a difficult skill to master, but it's a valuable one. To start your learning process, I recommend checking out *Manage Your Project Portfolio: Increase Your Capacity and Finish More Projects* by Johanna Rothman.^b It's one of the most useful books I've found on the subject of project management.

a. <https://archive.org/details/redeploy2018-failure>

b. <https://www.jrothman.com/books/manage-your-project-portfolio-increase-your-capacity-and-finish-more-projects/>

Decide Whether to End Project Engagement

Your organization, for many reasons, might consider ending its engagement in a FOSS project that it released. These can include the following:

- Not seeing expected return on investment (ROI)
- A change in strategic direction
- A change in another constraint, usually budget or staff availability

As soon as you suspect the company may need to withdraw from the project, start researching this option. The longer you wait, the more likely it is that you're creating a zombie that's going to devour any goodwill and benefits your company has gained from releasing the project. Consult the goals and strategies, and run the appropriate numbers. Does everything add up and align to support continued engagement with the project? If not, it's time to start the process of withdrawing from it.

Withdraw from the Project

Be open, honest, and transparent with the project's community. They don't necessarily need to know the details that led to the organization deciding to withdraw from the project, but they *do* need to know that it's withdrawing.

At all times, but especially in potentially difficult situations, the communication strategy with the community should operate on the Principle of Least Surprise.¹³ Don't make a withdrawal plan inside of the company and then pop it on the community at the last moment. Give them advance notice so they can help make the transition as smooth as possible.

Part of that transition is to see whether anyone else would like to step up to take over the company's leadership position within the project. Not only is asking for a new project steward the responsible thing to do, but locating that steward also relieves a lot of pressure on the company. The community will be able to see that they'll be in good hands.

Once you've located a new steward, collaborate with them on a leadership handoff and plan for your organization to transition away from its project responsibilities. Depending upon its constraints, your company may decide to stay involved in the project in a limited capacity, but it must still communicate and transition those responsibilities to ensure project and community continuity.

Close the Project

What if your company asks for a new steward, but no one's interested? Does that mean your company is doomed to stay shackled to this zombie forever? No, it doesn't. It can instead choose to close the project.

In this situation, the organization is responsible for coming up with the entire withdrawal transition plan. It should still communicate transparently with the community members, but as there's no new steward with which to collaborate, the organization can work within its own timeline. That timeline, though, should not be "immediately." It's a *transition* plan, which means it'll take some time. For the sake of the community and of your company's reputation, try to make that transition as smooth as possible.

The final "closing" of the project is actually more of an archiving. In fact, some forges like GitHub include a feature to archive a project in cases like this.¹⁴ Regardless of whether the project's forge has such a feature, you should explicitly state within the project documentation that it has effectively reached its end-of-life. Make this clear in the project's README file as well as its contribution guide and any other documentation or files that make sense for the project.

13. https://en.wikipedia.org/wiki/Principle_of_least_astonishment

14. <https://docs.github.com/en/repositories/archiving-a-github-repository/archiving-repositories>

Whatever else, do *not* delete the project from the forge or otherwise remove it from public access. It's likely that people are still using that project. I guarantee you that if they are and you take the project away, those people will soon be saying rude (and perhaps justified) things about your organization, demanding the project's return.

Also, preserving the project as available but unmaintained leaves open the possibility of someone revitalizing it in the future, usually by way of a fork. Should this happen, be empathetic, gracious, and supportive of those who wish to revive or take over the project. The project is under a FOSS license, after all, which implies that your organization wishes others to have access to inspect, modify, and redistribute it for their own purposes. You can hardly fault them for taking you up on that offer.

With this information, in combination with the books in [Appendix 5, Other Useful Resources, on page 423](#), you're now well on your way to maintaining a healthy and happy FOSS project and community. Now it's time to put all of your newfound FOSS knowledge to work by incorporating it into a business model.

Incorporate FOSS into Business Models

Throughout this book, you've learned that free and open source is a powerful tool in your business strategy toolbox. But we've yet to talk directly about how FOSS can help your business make money. That changes now.

In this chapter, you'll learn how FOSS can be a key element in a successful business model. You won't learn, however, about how to develop a business model in general. That topic is too large and is out of scope for this book, but I'll provide some basics here to get you started and to make sure we're all on the same page. If you need details about business model development, have a look at the book *Business Model Generation* by Osterwalder and Pigneur.¹

What Is a Business Model, Anyway?

Once in a while you find a term that's practically self-defining. Such is the case with *business model*, since it's the model of how the organization operates its business. Sounds simple, right? Alas, in business these things are never that easy.

"Business model" is a little phrase that hides an immense amount of complexity. That complexity reveals itself in the answers to the three questions that every good business model should answer:

1. *What is the value that the business is providing to consumers?* To answer this question, your organization needs to discover not only who your consumers are and what is of value to them but also which of those values your firm can help support.
2. *How will the business deliver that value?* Having a value for consumers means little if you can't get it to them, making it vital for the organization

1. <https://www.strategyzer.com/library/business-model-generation>

to consider how it will raise the awareness of its value offering (marketing) and how consumers can take them up on that offer (product, distribution).

3. *How will that delivery support the business goals?* It doesn't make much sense to put a lot of work into providing a value that doesn't itself help the organization meet its own goals, whatever those may be (often it's profit).

These are big, meaning-of-life-type questions for the organization—the *raison d'être* for the entire business—the answers to which comprise the business model.

Business Models Aren't Exclusive to For-Profits

As you likely guessed from that third question, even nonprofit organizations (NPOs) can and do have business models. For instance, let's consider the Humanitarian OpenStreetMap Team (HOT).² HOT is an international team dedicated to humanitarian action and community development through open mapping. The summary of its business model might look like this:

1. *What is the value that HOT is providing to consumers?* HOT provides "...map data that revolutionizes disaster management, reduces risks, and contributes to achievement of the Sustainable Development Goals."
2. *How will HOT deliver that value?* "HOT develops open source apps and tools for collaborative mapping and geospatial data collection. These tools are free for all to use..."
3. *How will that delivery support HOT's mission?* By providing easy, reliable, proactive, and trustworthy access to mapping data to humanitarian NPOs and NGOs, HOT minimizes the loss of life and property in crises and supports the creation and institution of policies that advance progress toward the United Nations Sustainable Development Goals.³

Of course the actual business model for HOT will be considerably more detailed, but this is enough to demonstrate that business models are still in place even where revenue and profits aren't involved.

A Business Model Is Not a Business Plan

It's common to use the phrases "business model" and "business plan" interchangeably, despite the two representing different facets of operating an organization.

2. <https://www.hotosm.org>

3. <https://www.un.org/sustainabledevelopment/sustainable-development-goals/>

In fact, a business model is only a single component of an overall business plan. That plan is a document that's used for defining goals for the organization and often is required for obtaining financing. Beyond a summary of the business model, the plan includes additional details such as market and competitive research, a marketing plan, a financial forecast, and descriptions of any other relevant elements of operation for the business.

Most of the time it's OK to use one of these terms in casual conversation when you mean the other, but it's never OK to forget that in practice they mean different things.

Common Elements of a Business Model

When you assemble a LEGO model of a sailing ship, you expect to see various bits and pieces, such as a hull, a mast and sail, and a rudder. Without these things, the model can't possibly be an accurate representation of a functional ship.

Business models similarly require certain elements to describe how you expect to operate and be successful with the resulting business. The elements required can depend somewhat on the business in which you're engaged, but usually include the following:

- Value proposition to deliver to customers and the differentiating qualities of that proposition
- Core capabilities that will allow the business to deliver value
- Target markets and the channels by which the business will reach them
- Cost structure and projections for all product and business operations
- Revenue model showing how the value will bring in enough money both to cover costs and to generate a profit

As you'll learn shortly, depending upon the business, FOSS can find a comfortable place in nearly every element of a business model. However, it's important to ensure that FOSS is there for a practical reason and not simply to chase trends. Adding impractical components to a business model leads to an unstable business and a lot of headaches.

Common Types of Business Models

You'll find at least as many different business models as there are businesses.⁴ The combination of all of the elements of a business model makes each model unique, but sometimes we can group a bunch of them together into

4. https://en.wikipedia.org/wiki/Business_model#Examples

categories or types to make it easier to talk about them. Here are a few types of business models with which you're likely already familiar:

Type	Example
Direct sales	Vendor at a farmers market
Commission	App store
Subscription	Newspaper
Ad-supported	Broadcast TV
Retail	Coffee shop
Fee-for-service	Plumber
Freemium	Open core software

It's not always possible to categorize a business model this easily, and that can be OK. After all, these categories are for convenience; no rule says you need to use them at all. However, if you come across (or generate) a business model that's difficult to categorize, have a good look at it. Either it's a model that's unnecessarily complex and therefore doomed to failure, or you may have something truly innovative on your hands.

Open Source Is Not a Business Model

If you hang around software business circles, it won't be long before you hear someone talking about a company having an "open source business model." That phrase is as nonsensical as a company claiming that their CEO is a manatee named Hugh.

Free and open source software (FOSS) is many things. It's a movement for free and shared access to resources; it's a community of practitioners; it's a collaborative development method; it's a critical component of business models. Therein lies the confusion: it's a *component*, not the model itself.

As you just learned, business models require several different elements for practical representation of a business. That model defines how a business meets its goals. For most businesses, that goal is profit. No direct profit is found in the free (in all English senses of the word) use, inspection, modification, and distribution of the differentiating value proposition.

When people talk about an "open source business model," usually they're referring to [Open Core, on page 381](#), which you'll learn about later in this chapter. It also could refer to other types of business models where FOSS may be a critical and visible component. Once in a while you find someone

using the phrase to refer to the fact that their firm does all of their planning and strategy in the open where anyone can see it.

In this way, confusing a component for the whole has made it difficult to recognize and discuss the many different ways that using FOSS in a business model can be a powerful strategic move. People see a visible FOSS component and don't look further to see the bigger model in which it resides. This self-inflicted limitation impedes the out-of-box thinking necessary to innovate on new FOSS-supported business models.

Whenever you're faced with someone talking about an "open source business model," take some time to learn what they mean by that. I guarantee it's not that the value proposition of the business is giving something away for free and expecting it to earn them a profit.

Open Source Can Have a Business Model

Many FOSS maintainers are eager to be paid for their work, and who can blame them? They and the projects they work on provide a great deal of value. What most of these people don't realize, though, is that getting paid either means working for someone or—more often—running a small business of your own. Even if it's only a one-person business, and even if the business is developing an openly licensed project, it has a better chance of success if a business model and a business plan are in place.

A business model represents how a business will create, deliver, and capture the value it provides to customers. FOSS projects already have the create and deliver steps sorted out (though there's always room for improvement in any endeavor). If you're someone who wishes to make a living out of developing FOSS, you'll need to fill in that final gap in the model: figuring out how to capture the value, aka how to get paid.

To do that, you'll need to discover and define exactly what it is your customers value with regard to the project. Think about it from their perspective. Talk to them. Once you have a firm grasp on the value the project provides to them, you can start to consider how to get paid for it.

You'll also need to run some financial numbers to determine exactly how much money you need (or want) to make. You may find that you'll need multiple revenue streams to come close to meeting that goal. Perhaps you'll have one revenue stream providing support to organizations that use the project, another for doing custom development, and a third for creating and delivering workshops about the project. If this is starting to sound like a lot of work, then you're getting the right picture; running a business isn't easy.

Don't forget that you may not be in this alone. Unless it's a single-maintainer project, a community of people are likely contributing to it. How do the business model and plan incorporate these people? If you're running a single-proprietor business around a multi-contributor project, you should always let the community know what you'll be doing. Communication can help prevent them feeling like you're exploiting and profiting from their work.

Making FOSS a Part of Your Business Model

How can you incorporate FOSS as a component of a business model for your organization? This is the last chapter and one of the last times you'll have to read my usual answer: it depends. As you learned in [Common Elements of a Business Model, on page 371](#), there are a lot of variables to consider when generating a business model, so each business model ends up being unique. Let's go through the list of model elements from that section and look at some examples of where FOSS might make sense.

Value Proposition and Its Differentiating Qualities

What is the value your business aims to provide to its customers, and how is it different—and ideally better—than what those customers can get elsewhere? This value must be something that the customers care about to some extent, not simply some “cool idea” the organization came up with and tries to sell to them.

- Is it important to the customers that they be able to switch vendors quickly and with minimal fuss rather than being locked into a contract for a proprietary provider? Then build your product offerings around FOSS-based projects and open standards, using “no vendor lock-in!” as part of your marketing messaging and sales pitch.
- Does the customer need the ability to modify and customize the product offering or audit the product for security or compliance reasons? Free and open-based software solutions give them access to the source code, even if in the end it's more cost effective for them to pay your organization to make the customizations.
- Is a certain FOSS project powerful but a right pain to install and operate? A business model pivoting around providing that project as a managed service or cloud-based software-as-a-service (SaaS) offering can provide a great deal of value to customers who can't (or don't want to) manage that project themselves.

Target Markets and Channels

Who are your customers and how will you reach them? You probably had to figure out the “who” early on so you could learn about their needs, constraints, and values. Now you can build upon that information to determine how best to speak with and reach the market. Targeting specific markets allows the business to focus its efforts rather than flail, trying to be everything to everybody.

- Small business owners need IT solutions too. Targeting a single vertical—real estate agents, for instance—with a package of FOSS-based services customized to their problem space limits the scope not only of the initial business but also of the potential channels. This makes the problems of that space easier to manage and manage well.
- Highly regulated industries, such as healthcare, have special requirements with regard to security and compliance but may not have the in-house skills necessary to maintain their FOSS supply chain. Developing a SaaS offering of complex software, such as Open Source Review Toolkit, and tailoring it for the needs of that market can help secure the customers who keep your friends and family healthy.⁵

Cost Structure and Projections

You already saw several examples of how FOSS solutions can contribute to reducing costs in [Lower Expenditures Through Intentional Use of FOSS Alternatives at Bees4Less, on page 136](#), where Bees4Less accomplished the following:

- Eliminated the need to maintain a fragile piece of software
- Improved onboarding times and staff productivity
- Converted the capital expenditure (capex) of server hardware to the operating expenditure (opex) of on-demand cloud computing

Starting out a business by choosing FOSS solutions, based on open standards, instantly changes the cost structure (usually for the better). The business will still need to operate and maintain those solutions, which requires skilled help, but it won't be locked into expensive, proprietary contracts.

Revenue Model

How can your business earn revenue—and ideally profit—from providing the value to customers? With apologies to South Park,⁶ it's a lot harder than (1)

5. <https://oss-review-toolkit.org/ort/>

6. [https://en.wikipedia.org/wiki/Gnomes_\(South_Park\)](https://en.wikipedia.org/wiki/Gnomes_(South_Park))

use FOSS, (2) ..., (3) profit. It's not as simple as slapping a price tag on a product and watching the money roll in. Factors such as cost structure and customer price sensitivity, among others, can have an outsized influence over profit margin.

- Offering a FOSS-based SaaS can provide a recurring and moderately stable income for a business.
- Building upon that SaaS by adding a professional services revenue stream to the model may allow for a larger margin.
- White-labeling the offering opens the possibility of strategic partnerships and revenue sharing.

However you choose to incorporate FOSS into a business model, make sure that all of the elements of that model mesh well. Like a clock that's missing a gear, a business model won't function well if the parts don't fit right.

Also, don't make the mistake of hanging all business hopes on FOSS. No single element of any model is strong enough to support an entire business. Build a strong business model where all the elements work together to create a stable and resilient business structure and enable the organization to meet its goals, whatever they may be.

Sell FOSS Products

Some folks may claim that it's "wrong" or "selling out" to put a price tag on openly licensed works. Don't listen to them.

You may recall from [What Is Free Software?, on page 5](#), that from its very inception, people have been encouraged to make money from free and open source software. Nothing in any OSI-approved license prevents it. In fact, article six of the Open Source Definition—No Discrimination Against Fields of Endeavor—specifically says that an open source license *must* allow people to do with the work whatever they will.⁷

You may find that you'll need to change the business's sales and marketing strategy to support selling a FOSS product. The same ol' playbook might not work. Or maybe it will? Determine what's valuable to your target market and speak to *that*. In some cases, the market may not care about whether the product is open or not, so if it's not a differentiator, then there may be no need to mention it.

7. <https://opensource.org/osd>

Consider the case of selling creative works available under Creative Commons licenses.⁸ I could, for instance, write a book of fairy tales and make it available under the CC BY-SA 4.0 International license. Then I could sell that as printed books, placing them in airport bookshops. Do the people buying the book care whether it's CC-licensed? No, they just want a diverting read for their flight home. The CC BY-SA licensing is important to me, as the author, but the market couldn't care less.

Because of that license, someone else could come along, translate the book to Spanish, and sell it themselves to a Spanish-speaking market. That's totally fine! The license allows it, I'm not going to sell to that market, and everyone is a winner. However, the license also requires that they give me credit for the original work and also their translation be licensed CC BY-SA, which gives the book the potential for endless variations and a long and free lifespan.

Selling a FOSS-based product isn't really so much different than selling a proprietary one. You still need to think through things like customer value, target market, competitive landscape, and economic environment. Obviously, selling a product that's under a FOSS license can come with additional variables that you'd need to consider before product launch, but that's why you're creating a business model in the first place.

FOSS Business Success Stories

Over the decades, many companies have intentionally included FOSS as a component of their business models. Some have even made FOSS *the* component around which everything else revolves and been successful in the process. Here are a few inspirational FOSS businesses.

Red Hat

You can't talk about FOSS business without talking about Red Hat,⁹ so we'll start there. Incorporated under its current name in 1995, Red Hat grew to become a multibillion-dollar enterprise until acquired by IBM for \$34 billion in 2019, where it operates as an independent subsidiary.

For most of those years, the business centered around the Red Hat Linux¹⁰ and Red Hat Enterprise Linux¹¹ distributions. These distributions—as well

8. <https://creativecommons.org/share-your-work/cclicenses/>

9. <https://redhat.com>

10. https://en.wikipedia.org/wiki/Red_Hat_Linux

11. https://en.wikipedia.org/wiki/Red_Hat_Enterprise_Linux

as nearly every other piece of software Red Hat works on—are available for free under FOSS licenses, mostly some version of the GNU General Public License (GPL). This means anyone could download and use the software themselves, without having to pay anything for licensing fees. How, then, did Red Hat become such a successful enterprise?

Red Hat built its brand and earned the trust of its customers, who valued the stability and support that came along with a subscription for Red Hat–trademarked services. Red Hat software may have been freely available, but it was also professionally maintained and tested specifically for large enterprise customers. Even when Red Hat started its move to cloud computing, all its software remained FOSS-licensed. They developed these software components with high degrees of integration and interoperability, with their customers in mind.

Keeping their customers front-and-center and building trust in their trademark and brand is what made Red Hat the successful FOSS business juggernaut it is today.

GitLab

GitLab¹² is an open core online collaboration forge and DevOps¹³ company (you’ll learn this in [Open Core, on page 381](#)). In 2023, the company reported revenues in excess of \$420 million, even though the company’s software is freely available for anyone to download and install, and its SaaS version is free for all to use.¹⁴

The company is able to do this by understanding and meeting the needs of their customers. While anyone *can* download, install, and administer the software—and GitLab provides superb documentation¹⁵ to help them do this—many companies *don’t want to*. Why take on that burden when they can outsource it to the skilled employees of GitLab?

As a company, GitLab takes openness seriously. Not only is its company handbook available for anyone to view online,¹⁶ but also those readers can contribute to, copy, modify, and redistribute the handbook since it’s available under the highly permissive MIT FOSS License.¹⁷

12. <https://gitlab.com>

13. <https://en.wikipedia.org/wiki/DevOps>

14. <https://about.gitlab.com/install/ce-or-ee/>

15. <https://docs.gitlab.com>

16. <https://handbook.gitlab.com>

17. <https://gitlab.com/gitlab-com/content-sites/handbook/-/blob/main/LICENSE>

Grokability and Snipe-IT

Snipe-IT is asset management software created and operated by the Grokability company,¹⁸ and it's proud and loud about being Free Software available under the GNU Affero General Public License version 3.0 (AGPL-3.0-or-later).¹⁹

As an AGPL-licensed product, Snipe-IT is freely available for anyone to download, install, and operate on their own, and the Grokability team encourage you to do so. However, should you run into any problems with that installation, you won't receive priority support unless you've purchased it.²⁰ Alternatively, and much like GitLab, you can pay to have their team host and maintain a Snipe-IT installation for you. The subscription fee also entitles companies to request new features, which will be prioritized in the product roadmap.

Of all the example companies, Grokability is the one that I feel is most inspirational, as it shows the power of building a business model around a FOSS component. Unlike the other companies listed here, Grokability is entirely bootstrapped (self-funded) and independent. It has grown slowly and sustainably, is profitable, and has a reputation for excellent service both to its customers and to its employees.²¹ Prioritizing the needs of its customers in an open way generates value for which people are happy to pay.

It's More About Business Than FOSS

Despite the evident success of these and many other companies, we still find a number of influential business advisors who recommend avoiding business models with FOSS elements. They claim that these models hold too much risk. In one respect, these people are correct: risk is present in business models that incorporate FOSS, but that's because risk is present in *every* business model.

Failure abounds in the business world. According to the U.S. Bureau of Labor Statistics, 50 percent of new companies fail within five years of launch.²² Tom Eisenmann, author of *Why Startups Fail: A New Roadmap for Entrepreneurial Success*,²³ found that a full two-thirds of all companies never show a positive

18. <https://snipeitapp.com/company>

19. <https://snipeitapp.com/product/open-source>

20. <https://snipeitapp.com/support>

21. <https://snipeitapp.com/company>

22. https://www.bls.gov/bdm/us_age_naics_00_table7.txt

23. <https://hbr.org/2021/05/why-start-ups-fail>

return to investors. The reasons for these failures are legion but often can be summarized as “not understanding the business elements well enough to craft and operate a resilient business model.”

In the 2010s, the software world saw an explosion of startups that distributed their products under FOSS licenses while simultaneously operating revenue streams around SaaS offerings, support contracts, or subscription-based enterprise features. A great many of these startups imploded, unable to sustain their business models. Were these models inherently flawed because they included FOSS elements? No, they were not. You learned in [FOSS Business Success Stories, on page 377](#), that companies can find great success with FOSS-based business models.

Protective Business Model Elements

The problem wasn't that these business models were flawed but that the business operators likely didn't have adequate knowledge or experience to understand how to make the models work. Companies like Grokability, GitLab, and Red Hat intimately understand not only FOSS but also their customers and their needs. They're aware that someone else could easily take their software and launch a competing business, and they crafted their business models around this risk. These are some of the protective elements of their models:

- *Trademark:* All Red Hat software is protected by their trademark. You're free to take their software and create a new company using it, but you're prohibited from calling it or otherwise using any of the marks that they've registered. This prevents anyone from trading off the brand they've worked so hard to build.
- *Licensing:* Grokability's Snipe-IT software is available under the highly reciprocal AGPL network license, which allows anyone to take the software and run it however they wish, including to start a competing product offering. However, should the competitor make any modifications to the software—which they would need to do, if only to remove the trademarked Snipe-IT branding—then they're required to release the source of those modifications. Few companies are as knowledgeable or skillful with FOSS as Grokability such that they can build a successful business on software they didn't originally create. Furthermore, they'd need to overcome the strong first-mover advantage and brand recognition that Grokability has already earned.

- *Community investment:* As you learned in [Community, on page 104](#), community is the superpower of FOSS. Despite this, few companies with FOSS-including business models pay adequate (or any) attention to building and sustaining a strong community around the FOSS projects. GitLab understands community and invests a lot of time, talent, and treasure not only in the community around their own flagship project but also in the overall FOSS ecosystem. This brings brand recognition, yes, but it also brings *product* recognition. Community members remember the companies that are good to them and recommend those companies to their friends and colleagues.

Beware a Lack of Investment

Lack of investment, or misdirected investment, is a common mistake in all new businesses, but those with models including FOSS seem to be more susceptible than most. Again, this stems from a lack of understanding.

For years now, the greater FOSS community has rightfully waved the flag of collaboration, extolling the virtues that come with an open and transparent development model. This, combined with the lack of licensing fees required for project acquisition, has led many companies to neglect their FOSS components with the assumption that someone else (read: the community) will handle everything. As you learned in [The Situation with FOSS Sustainability, on page 217](#), this neglect can have disastrous results.

FOSS-including business models therefore require intentional investment in the component FOSS projects to have any hope of success. Not viewing the FOSS as the strategic element that it is and shortchanging it rather than applying the attention required to build a strong community leads to business failure in these cases.

FOSS-based businesses are not a spectator sport. You can't stand by the sidelines and hope the game goes your way. If FOSS is strategic enough to your business that it's a part of your business model, then you must get in the game by investing in your organization's FOSS engagement.

Open Core

Open core—sometimes called *commercial open source software* (COSS)—is a type of business model that's received a lot of attention in the past decade through to today. It's a specialized form of the *freemium* type of business

model,²⁴ where a company provides a limited version of a product at no cost in hopes of converting those free users into paying customers.

In open core, the limited product version that's provided is available under a FOSS license. If you recall, [GitLab, on page 378](#), operates under an open core business model. Anyone can download the free version of their eponymous product because it's permissively licensed with the MIT License. Open core companies then establish revenue models that provide additional value (also known as *value-adds*) to the customer. Some of GitLab's value-adds are additional storage space, compute power, and support options. Other common open core value-adds include advanced features and access to the product via a SaaS subscription.

Think Before You Leap into Open Core

Open core can be a powerful business model when done well, but it does require thought before getting started. Consider these things before releasing your business differentiator under a FOSS license:

- *Make sure you understand the values and needs of your target market.* For instance, if your market doesn't mind installing and maintaining their own installation of your software product, they won't be converted to paying customers by the value-add lure of a SaaS subscription. Make sure any value-adds actually add value for your potential customers.
- *Know the real and potential competitors in your space.* Are any of those companies likely to take advantage of your FOSS project to develop and launch a more compelling product offering? In the cloud-computing-based world we now live in, it's easier than ever for a competitor to do this. Make sure your business model makes provisions for this possibility, such as focusing on and capturing a specific target market or industry vertical.
- *Build a shield of people who love your company and its project.* In other words: build community. These people will protect the project from incursions from those they feel might damage or diminish the community, assuming your business has taken the time to engage and invest authentically in it.

Unfortunately, it's become trendy for software startups to use an open core business model despite not fully understanding the risks and, therefore, not taking any of these steps to minimize them. This has led to a number of

24. <https://en.wikipedia.org/wiki/Freemium>

companies reconsidering their business models and relicensing the “core” to a proprietary, source-available license from a FOSS one.

Relicensing Open Core

Relicensing happens more frequently in the past few years, mostly among (now formerly) open core companies. Sometimes, an open core company discovers that it can’t make its business model work well enough to meet company needs. Usually this results in the company—which had previously been giving a version of their software away as FOSS—changing its business model away from open source and relicensing the software as proprietary. They’re able to do this more easily because they required external contributors to the projects to sign CLAs.

CLAs Enable Easier Relicensing

Many open core companies encourage collaboration on and contributions to their FOSS projects, but they also usually require that those contributions only come after someone has signed a Contributor License Agreement (CLA).

As you learned in [Contributor License Agreements, on page 34](#), by signing a CLA, a project contributor affirms the following:

1. They have the right to share the IP that’s in the contribution.
2. They give the project license to alter, distribute, and administer their contribution.
3. The contributor will never revoke that license.

Legal mechanisms like a CLA come into play primarily in the case of changing a project’s license, something that ideally should not happen very often.

That second item in the list of CLA affirmations is especially important here. Changing the license under which that contribution is distributed is an alteration permitted with this affirmation. Those who have administrative powers over the project (the originating open core company) can then freely change the project’s license however they see fit, be that from one FOSS license to another or even from a FOSS license to a closed proprietary one.

Community Reaction to Relicensing: Fork It

To say contributors feel upset about these relicensings would be an understatement. They feel they contributed their time and skills for free, in the good faith that those contributions would also remain freely available. By relicensing to become proprietary, the company has therefore disrespected the community

and taken advantage of them. The open core company loses the trust of the community and can take a lot of damage to its brand.

When faced with such a change, the community tends to fork the final freely available version of the software to create a new community-managed project (as you learned in [Stick a Fork in It, on page 362](#)). This has happened several times in recent years, including in these cases:

- Elastic relicensed the Elastic and Kibana projects as proprietary,²⁵ leading to a fork and the creation of OpenSearch.²⁶
- HashiCorp relicensed Terraform as proprietary,²⁷ leading to a fork and the creation of OpenTofu.²⁸
- Redis relicensed the eponymous Redis project as proprietary,²⁹ leading to the creation of several forks, including redict³⁰ and Valkey.³¹

The keys to avoiding this sort of community reaction are the same as ensuring success in any undertaking: understand and communicate the goals, take the time to learn about the ecosystem and its potential pitfalls, and then plan and act collaboratively. By reading this book, you've taken a big step toward fantastic success with free and open source within your business.

25. <https://www.elastic.co/pricing/faq/licensing>

26. <https://opensearch.org>

27. <https://www.hashicorp.com/blog/hashicorp-adopts-business-source-license>

28. <https://opentofu.org>

29. <https://redis.com/blog/redis-adopts-dual-source-available-licensing/>

30. <https://redict.io>

31. <https://github.com/valkey-io/valkey>

Acronyms & Initialisms

API: application programming interface

ASF: the Apache Software Foundation

CAD: computer assisted design

CC: Creative Commons

CC0: Creative Commons Zero license

CCLA: Corporate Contributor License Agreement

CISA: Cybersecurity and Infrastructure Security Agency

CLA: Contributor License Agreement

CoC: code of conduct

COSO: chief open source officer

COSS: commercial open source software

CRA: Cyber Resilience Act (EU)

CSV: comma-separated values

CTO: chief technical officer

CVE: common vulnerabilities and exposures

CYA: cover your ass

DCO: Developer Certificate of Origin

DEI: diversity, equity, and inclusion

DFSG: Debian Free Software Guidelines

EF: the Eclipse Foundation

EPL: Eclipse Public License
EULA: end user license agreement
FLOSS (also written F/LOSS): free/libre and open source software
FOSS: free and open source software
FSF: Free Software Foundation
FSFE: Free Software Foundation Europe
FUD: fear, uncertainty, and doubt
GPL: GNU General Public License
GSG: Getting Started Guide
GSOC: Google Summer of Code
HOT: Humanitarian OpenStreetMap Team
IANAL: I am not a lawyer
IANYL: I am not your lawyer
ICLA: Individual Contributor License Agreement
IoT: Internet of Things
IP: intellectual property
IPO: initial public offering
LF: the Linux Foundation
LGPL: Lesser GNU General Public License
M&A: merger(s) and acquisition(s)
MPL: Mozilla Public License
MVP: minimal viable product
NBT: next big thing
NGO: nongovernmental organization
NIH: not invented here
NIST: National Institute of Standards and Technology
npm: Node(.js) package manager
NPO: nonprofit (not-for-profit) organization

NTIA: National Telecommunications and Information Administration

NVD: National Vulnerability Database

OCTO: Office of the CTO

ODC: Open Data Commons

OIN: Open Invention Network

OKFN: Open Knowledge Foundation

ORT: Open Source Review Toolkit

OSD: Open Source Definition

OSHWA: Open Source Hardware Association

OSI: Open Source Initiative

OSPO: Open Source Program Office

OSRB: Open Source Review Board

OSS: open source software

OWASP: Open Worldwide Application Security Project

P&P: policies and procedures

PD: public domain

PII: personally identifying information

PURL: package uniform resource locator

RMS: Richard M. Stallman

ROI: return on investment

SaaS: software as a service

SBOM: software bill of materials

SCA: software composition analysis

SCOTUS: Supreme Court of the United States

SDG: Sustainable Development Goal (from the United Nations)

SDLC: software development life cycle

SEO: search engine optimization

SFC: Software Freedom Conservancy

SLSA: Supply-Chain Levels for Software Artifacts

SPDX: System Package Data Exchange

SPI: Software in the Public Interest

SPOF: single point of failure

SRE: software reliability engineer(ing)

SSG: static site generator

SWID: software identification tags

TAM: total addressable market

TCO: total cost of ownership

TINLA: this is not legal advice

ToS: terms of service

TTM: time to market

VCS: version control system

WIPO: World Intellectual Property Organization

YAGNI: you ain't gonna need it

YOLO: you only live once

Glossary

Most of the words in this glossary have multiple meanings, or different ones depending upon the context in which you use a word. The definitions I provide here are all in the context of this book and its subject matter.

administrivia: Administrative or other types of work that people perceive as trivial or bothersome.

agile development: A methodology for (usually software) project development that operates according to the *Manifesto for Agile Software Development*, which prioritizes *individuals and interactions* over processes and tools, *working software* over comprehensive documentation, *customer collaboration* over contract negotiation, and *responding to change* over following a plan. Most “agile” teams today follow heavyweight, process- and tool-driven practices that have diverged from the original intent of the manifesto.

all rights reserved: A declaration by a copyright holder (author) that they retain all rights over the work. No one else is allowed to use or distribute the work without express permission from the author. The phrase used to be required on the work in many jurisdictions for the author to gain copyright protection for their work, but it’s now outdated.

anti-pattern: A commonly used pattern of operation or practice that appears at first to be a good idea but in fact has more negative effects than positive. Despite that, it remains in use.

application programming interface (API): A sort of vocabulary that programmers build into a software system that allows other systems to interface or communicate with it.

author: The creator of a work, with respect to intellectual property and copyright.

backport: Applying features or fixes from a newer version of software back onto an older version. Is especially common for security fixes.

bare license: Gives permissions to use a work in some specific way but shares no ownership or other rights over that work.

brand: The public image, perception, or reputation of an organization, good, service, or individual.

break the build: Contributing code or related errors that cause a software build process to fail in the compilation, testing, or other verification steps.

build: See *software build*.

capital expenditure (capex): A financial and accounting term for the money that's spent on fixed, long-term assets. Often mentioned in opposition to *operating expenditures (opex)*.

chaos engineering: The practice of testing and developing resilience in a production software system. Modern chaos engineering practices include running utilities that randomly inject failure states into a system to gauge how the system and its operators tolerate and recover from failures.

clone: To copy a repository of a version control system. The GitHub collaboration forge instead, and confusingly, uses the term *fork*, which has led to a confusion between the terms in common usage.

cloud: Computing resources that are available on-demand via the internet. Organizations use cloud services instead of buying or building their own computing resources and software, paying only for the services and resources they use.

cloud native: Designing and architecting software to make the most of operating in the cloud. Observability, resiliency, and automation are key elements of most cloud native software.

code of conduct: Rules and expressed expectations around the accepted norms of behavior in a community or organization.

cognitive load: The amount of mental capacity or resources used at a point in time. An increase in someone's cognitive load can lead to a decrease in their overall performance.

combined work: A work that's the result of combining two or more works that are available under different FOSS licenses. Some of these licenses have terms that aren't compatible, which would prevent the combination of works under incompatible licenses.

commercial open source software: See *open core*.

commit hook: A script or other software that automatically runs against a version control system (VCS) repository when someone uses the VCS software. Common commit hooks include pre-commit tests that catch common errors, stopping the commit before the errors can make it into the repository.

community-oriented enforcement: An approach to FOSS (usually reciprocal) license compliance that rejects punitive enforcement efforts in favor of collaborating with license violators to bring them into compliance, only resorting to legal action as a last resort should violators repeatedly refuse to cooperate and comply with the terms of the licenses.

compliance: Using software or other licensed works in accordance to the terms and conditions established by the copyright holders and defined in the license.

coopetition: A portmanteau of “cooperation” and “competition,” it’s the practice of cooperating or collaborating among otherwise competing people or organizations.

container: A type of software package that collects up everything a piece of software needs for operation. This isolates it from the underlying operating system and other software, meaning, theoretically, that the software should operate the same wherever you deploy it.

container image: An unchangeable snapshot of a container that allows it to be shared or deployed consistently.

container registry: An online catalog for storing and accessing container images.

copyleft: See *reciprocal license*.

copyright: The right to control the copying, use, repurposing, or redistribution of a work.

copyright assignment agreement: A legal agreement for assigning or transferring copyrights for a work from one legal entity to another.

core contributors: FOSS community members who have shown a continuing commitment to the project and its community as well contributing their time and skills. These people usually receive increased responsibilities and rights in the project, often including being able to merge or reject contribution requests and having increased influence over the project roadmap.

CRA: See *Cyber Resilience Act*.

crayon license: A derogatory term sometimes applied to licenses. It implies that the license is the work of someone who isn't familiar with intellectual property law or the norms and expectations of license drafting. The license author is an amateur who may as well have written the license in crayon.

CVE: Acronym for Common Vulnerabilities and Exposures, a system for tracking and announcing software security vulnerabilities.

Cyber Resilience Act: The EU Cyber Resilience Act (CRA), initially proposed in 2022 and still awaiting formal adoption at this time in 2024, intends to establish common cybersecurity standards across the European Union.

CycloneDX: A bill of materials (BOM) standard, supporting multiple different types of BOMs including software (SBOM), hardware (HBOM), and manufacturing (MBOM).

derivative work: A modification to an existing copyrighted work, or a work that incorporates some or all of another copyrighted work.

DevOps: A practice in software development that prioritizes communication, collaboration, and shared ownership for deliverables. A portmanteau of “developer” and “operations,” DevOps reduces or eliminates the silos between these two roles in software development, allowing for more rapid and reliable software delivery.

DevRel: An abbreviation for *developer relations*, which is a function that benefits the organization by building relationships and trust in the greater external software developer community. This provides valuable marketing insight, product feedback and direction, customer satisfaction, and community building.

distribution: With regard to licensing, it's selling or giving away that software to anyone outside of your immediate company or organization. If the software your company has created is a derivative of any FOSS projects, and if your company then sells or gives away that software to customers, partners, or even to random strangers on the street, then it's distributed the software. See also *Linux distribution*.

distro: Abbreviation of *distribution*. Usually used in the context of a Linux distribution.

domain name: A usually human-readable identifier that directs to a resource on the internet such as a website, an email server, or an application. Domain names must follow a standardized format and be registered to be publicly accessible.

downstream: Any use of or activity with a software project that is done away from its authors. See also *upstream*.

dynamic linking: Loading software dependencies at the time the software runs rather than compiling them in to become a part of the software itself. See also *static linking*.

end user license agreement (EULA): A legal contract to which a person must agree before they may use a piece of software. The EULA details the terms, conditions, rights, and restrictions that apply when someone chooses to use the software.

EO 14028: Abbreviation for *United States Executive Order 14028: Improving the Nation's Cybersecurity*. This order calls for many specific steps that organizations must take to improve the cybersecurity of various systems, including providing SBOMs for the sake of dependency transparency and awareness.

fair dealing: See *fair use*.

fair use: A concept in copyright law that acknowledges that sometimes it's OK for members of the public to use portions of a copyrighted work for certain purposes, even if they don't have permission from the copyright owner.

fauxpen source: A derogatory term used in reference to some source-available software, especially that which previously had been available under a FOSS license.

fiscal sponsor: A nonprofit that provides administrative assistance to external projects that align with the sponsor's mission. The sponsored project then gains all of the advantages of having a nonprofit status without needing to form a new nonprofit itself. Some fiscal sponsors in the FOSS ecosystem include Software Freedom Conservancy (SFC) and Software in the Public Interest (SPI).

fork: To take a copy of a FOSS project, rename it, and start a new project and community around that copy. Those who fork a project rarely, if ever, contribute to the parent project again. While the fork does require some technical work, it's primarily a social action.

forward port: To apply features or fixes from an older version of software forward onto a newer version. Is especially common for security fixes. Not to be confused with the internet networking concept *port forwarding*. See also *backport*.

Four Freedoms: The core of the Free Software movement. Enumerated from zero to three, these freedoms are: the freedom to run a program as you wish, for any purpose; the freedom to study how the program works and change it so it does your computing as you wish; the freedom to redistribute copies so you can help others; and the freedom to distribute copies of your modified versions to others. The Four Freedoms often are abbreviated as “the freedom to use, inspect, modify, and redistribute.” Not to be confused with the *Four Freedoms* expressed by President F. D. Roosevelt in 1941.

free as in beer: A concept in free and open source software, related to the many different meanings of the word *free* in the English language. When people in FOSS want to say that something costs no money, they say it’s “free as in beer.” See also: *free as in freedom*, *free as in mattress*, and *free as in puppy*.

free as in freedom: A concept in free and open source software, related to the many different meanings of the word *free* in the English language. When people in FOSS want to say that something provides or sustains the freedom of users to use, inspect, modify, and redistribute the software, they say it’s “free as in freedom.” See also: *free as in beer*, *free as in mattress*, and *free as in puppy*.

free as in mattress: A concept in free and open source software, playing off the many different meanings of the word *free* in the English language. When people in FOSS want to say that something may provide the Four Freedoms (free as in freedom) and cost nothing to acquire (free as in beer), but it also may be riddled with bugs, toxic, and generally unpleasant, they say it’s “free as in mattress.” See also: *free as in beer*, *free as in freedom*, and *free as in puppy*.

free as in puppy: A concept in free and open source software, playing off the many different meanings of the word *free* in the English language. When people in FOSS want to say that something may provide the Four Freedoms (free as in freedom) and cost nothing to acquire (free as in beer), but it also requires people to maintain it and give it attention to stay healthy, they say it’s “free as in puppy.” See also: *free as in beer*, *free as in freedom*, and *free as in mattress*.

Free Software: A social movement and associated software community and ecosystem built around the Four Freedoms. Free Software is the progenitor and sibling of open source, with which it coexists and collaborates. In many contexts, Free Software and open source can be treated as a single movement. See also *Four Freedoms*; *open source*.

freemium: A type of business model where a business provides a limited version of a product at no cost in hopes of converting those free users into paying customers.

Git: A popular version control system. It's typically used as a fundamental tool in software development, allowing multiple people to work on the project at once and tracking changes, but it's also useful for other types of projects (I used it for this book!).

hack: A slang term for writing or working on different elements of a software project.

hackathon: An event—online or in-person—for people to gather and work on a software project in a focused and usually collaborative way.

hallway track: The informal networking and socialization that occurs at conferences and other professional events. The name refers to the fact that these conversations are emergent as people meet each other in the hallway between event sessions.

inbound equals outbound: A method of intellectual property management for contributions to FOSS projects where the license under which the contributor provides their contribution (inbound) is the same as that of the project itself (outbound).

infosec: An abbreviation for *information security*, which is the practice of protecting critical and confidential information.

infringement: To ignore or otherwise violate the terms of a license.

internal fork: A fork of a FOSS project that's solely for use internally within an organization and will never be distributed. Often internal forks happen because a business requires changes to a project but doesn't want to contribute those changes back upstream. Long-running internal forks are an anti-pattern you should avoid.

Internet of Things (IoT): Equipping otherwise nonnetworked devices and objects with networking and computing capability, allowing internet-based collection, sharing, and use of data by those objects.

jurisdiction: The area or region where certain laws and authority can be enforced.

kernel: The software that controls all fundamental operations on a computer, including memory management, hardware management, and process management. Linux is a popular kernel, but it's not the only one. The BSD kernel—at

the core of operating systems such as the popular FreeBSD—is another. UNIX, Windows, and macOS all also have their own versions of a kernel.

killer app: A software application or other technology with features and benefits that make it a must-have app. Often a killer app is required to drive adoption of a new technology.

land a patch: To have a FOSS project accept a contribution.

legal person: A person or other entity that can enter into contracts, own property, sue or be sued, or perform other legal actions. Companies usually count as legal persons, but the qualifications for this personhood vary by jurisdiction.

license: Declares what can and can't be done with a copyrighted work or patent, by whom, and under what conditions. Because they're giving permissions for intellectual property, licenses are legal instruments that are enforceable by law.

Linux distribution: A collection of utilities and other software packaged together along with a kernel as a complete and functional operating system. The software is curated and packaged to maximize standardization and interoperability. Linux is the kernel in the case of a Linux distribution. Some popular Linux distributions—often called “distros”—are Debian, Fedora, and Ubuntu.

liquidity event: A business event or operation that allows investors in an enterprise to get some return on that investment. An initial public offering (IPO) and merger/acquisition (M&A) are types of liquidity events.

load-bearing links: Links in your software supply chain, the loss or failure of which would cause a crisis in your business and its operations.

loss leader: A product or other offering that's provided at a low price or for free to attract customers who might then buy higher-priced products.

Markdown: A lightweight, human-readable markup language for text, commonly used for documentation and other text in FOSS projects. This book was written in Markdown! (with some extra post-processing to make it pretty).

mere aggregation: A concept in intellectual property law that says if two or more things are collected together but do not interact in any way, they don't form a single work. You can't patent a collection that's a mere aggregation, and if that collection includes components that are reciprocally licensed, then the license doesn't apply to the other noninteracting components.

minimum viable product (MVP): A product or service that contains just enough of the anticipated functionality that it can provide value to its consumers, who can then use it as a basis for providing feedback to shape future development.

notices: The copyright and license attributions for the components of a FOSS project.

open core: A specialized form of the freemium business model where a business provides a limited and FOSS-licensed version of a product in hopes of converting its users into paying customers.

open source: Copyrightable works released under licenses that the Open Source Initiative (OSI) has reviewed and certified as complying with all ten parts of the Open Source Definition (OSD). Also commonly and confusingly applied to anything that's freely available for the public to use, inspect, and modify, regardless of whether it complies with the OSD.

Open Source Program Office (OSPO): A cross-functional team to help define and steer an organization's open source management strategy and organizational readiness. *Definition courtesy of the OSPO Alliance.*

open standard: A standard that's developed in the open, such that anyone can participate or follow along in its creation, and then is publicly available for anyone to use. Open standards typically are released under open licenses, such as those from Creative Commons.

open washing: Deceptively presenting something as open and freely available when in fact it's limited in some way. Usually the deception is intentional, but sometimes it's due to a misunderstanding of the community's expectations of what qualifies as open.

operating expense (opex): An ongoing expense related to the operation of a business or undertaking. Consumable materials, rent, and similar expenses can qualify as opex.

package uniform resource locator (PURL): A URL string used to identify and locate a software package in a mostly universal and uniform way across programming languages, package managers, packaging conventions, tools, APIs and databases. Used when maintaining a software supply chain.

patch: A contribution—usually a bug fix—to a FOSS project. Also can be used as a verb when applying that contribution: “Don't forget to patch that security vulnerability in version 3.2.”

patent pool: A collaboration among patent holders. All members of the pool agree that other members are allowed to use the patents they place in the pool without fear of legal or financial action. Through this collective action of the members, they form a coalition that holds off patent trolls.

patent troll: Unsavory individuals and corporations who buy patent portfolios, typically of small or failing businesses, and then use threat of patent infringement lawsuits to extort licensing fees from others.

pay-to-play: Paying a monetary fee to participate in or contribute to an ostensibly FOSS project. See also *open washing*.

permissive license: These licenses permit people to do different things with a project than a reciprocal license would. Whereas works that incorporate reciprocally licensed FOSS projects must be released under licenses that retain that reciprocity forever, works incorporating FOSS projects released under permissive licenses can do whatever they like with their final work as long as they meet the license conditions.

pinning: Restricting use of a FOSS software component to a specific version to aid in maintaining predictable and stable functionality. Can increase risk if not properly maintained, since the pinned versions may contain security vulnerabilities that have been fixed upstream.

premature optimization: Putting in a lot of effort to improve the performance of something that doesn't really need it. Either the thing is rarely used or the improvement is so marginal that the effort isn't worth it. Software programmers at all levels of skill commonly spend a lot of time on applying well-intentioned but expensive premature optimizations in their code, to the point where feedback about it has a special acronym: YAGNI (see *YAGNI*).

prior art: A concept in patent law. You can only patent new things, aka *novel inventions*. If something already exists that's like your invention, it's *prior art*, and therefore your invention is not novel and you cannot receive a patent for it.

product stickiness: See *sticky product*.

proprietary license: A license that does not comply with the Open Source Definition and therefore cannot be certified to be *open source*. The overwhelming majority of licenses in the world are proprietary. Nothing is inherently wrong with them; it's simply a way to categorize and differentiate them from OSI-approved FOSS licenses.

README: The primary entry point to a FOSS project. A README file should include the purpose of the project and where to find other important information, such as usage and installation guides, how to contribute to the project, and the project's code of conduct.

reciprocal license: Software that is based upon or incorporates FOSS projects that are under reciprocal licenses *must* be released under those same permissions and conditions. This is where the reciprocity comes in: as you have benefitted from this FOSS project, so must others similarly benefit from your own work if it incorporates the project. The permissions and conditions of these licenses are rather like the software IP equivalent of the Golden Rule: do unto others as you would have them do unto you—or as you have done, thanks to them. You've received and used the reciprocal software under certain terms and must ensure that others can use all derivative software under the same terms.

risk profile: The characteristics of an individual or organization, the evaluation of which allows them to gauge how much risk they're able to take. Things that can affect a risk profile include financial health and liquidity, laws and regulations, and potential costs and impacts should the risk not work out. Companies in highly regulated industries have a low risk profile (must be especially careful when taking on risks), as are people who have the sole income in their family.

role address: An email address that's formatted and tied to a role rather than an individual, for instance `ospo-licensing@bees4less.com` rather than the individual who performs the role of managing FOSS licensing for the Bees4Less OSPO. Use and proper management of role addresses rather than individual addresses reduces the risk of important emails going astray as people move on from a company or role as well as the need to update email addresses in policies, procedures, and other documentation.

roll your own: Creating your own solution when alternatives already exist. This is a common tobacco/cannabis term that technologists have adopted.

search engine optimization (SEO): Manipulating the metadata and content of a web page to improve the page's ranking in search engines, therefore also increasing the visitors that arrive at the page from clicking its entry in search engine results.

self-documenting: Software or other projects that use naming schemes and other established conventions and best practices to make it as easy as possible for a new user or collaborator to understand the purpose (but not necessarily the rationale) for each segment of the project.

server (hardware): A computing device that provides access to applications, files, or devices. These may be available over the network or for local use only.

server (software): Software that provides networked access to resources such as web pages, web applications, and similar things. Servers (software) usually run on servers (hardware), which in conversation sometimes gets just as confusing as you'd expect.

service mark: Trademarks, but for services offered to customers. While there's a difference between a service mark and a trademark, in common practice the word *trademark* usually covers both terms.

shadow IT: Software or hardware that a team or an individual uses without the knowledge or approval of the IT department. FOSS applications and components often find their way into an organization by way of shadow IT.

shift left: The concept that things get a lot harder and more expensive to fix or change later on in the timeline of a process, so it's better to handle real or potential problems earlier on—to “shift left” on the timeline.

single-vendor commercial FOSS project: A FOSS project that has its development and maintenance dominated or entirely controlled by a single vendor. Typically the project forms the core of the vendor's open core business model.

snippet: A small piece of software source code that can be or is reused in another piece of software. A snippet can complicate a copyright and licensing compliance program because it's rare for it to include the information necessary for the compliance, yet some entity still has copyright over the snippet, and it may be available under some license. Some SCA tools include tests to locate snippets, but usually these locate a flood of false positives, diminishing or even eliminating the usefulness of that functionality.

software as a service (SaaS): Providing an application or other software-based service over the internet. SaaS usually operates on cloud computing infrastructure and platforms.

software bill of materials (SBOM): A document that provides an inventory of a piece of software, its components, and dependencies. Thanks to the complexities of modern software, SBOMs can be long, and in cases of poor software supply chain practices, they can be difficult to create.

software build: Either the process of running automated tests, integrations, and compilations for a piece of software or software that results from the process. If the process fails for an unexpected reason, it's said to be broken. See also *break the build*.

software composition analysis (SCA): The process of analyzing a piece of software to determine various data about it, including its dependencies (components), the licenses associated with each component, the version of the component, and any security concerns or vulnerabilities reported for each component.

software identification (SWID) tags: An ISO standard (19770-2) for metadata describing a piece of software. SWID tags are commonly used in software asset management and software supply chain management. Some SCA tools use SWID tags.

System Package Data Exchange (SPDX): An open international standard (ISO/IEC 5962:2021) actively developed under the Linux Foundation. It defines a format, readable by both humans and computers, for SBOM documents. Initially focused on gathering and communicating data necessary for copyright and license compliance, over the decade-plus of its existence SPDX has grown to include all necessary data for a software supply chain as well as supporting AI, data, and security use cases. Formerly known as Software Package Data Exchange.

source available: A term used to describe software that has publicly accessible source code, but that code is not available under an OSI-approved FOSS license.

SPDX: See *System Package Data Exchange*.

static linking: Loading software dependencies at the time the software is compiled, making those dependencies a part of the software itself. See also *dynamic linking*.

static site generator: Software that assembles templates and text-based content written in human-readable markup languages (such as Markdown) and uses those to generate static HTML files. Statically generated websites usually are very lightweight and load quickly in web browsers. It's very common for FOSS projects to have websites created by a static site generator.

sticky product: A product or service that customers find it difficult to switch away from due to features they love, a brand they trust, or value that they don't wish to give up.

systems thinking: A holistic overview of complex and intertwined components that make up a system, including all component interactions and resulting effects. Often people refer to *big-picture* or *fifty-thousand-foot* views when talking about systems thinking.

technical debt: Backlogged housekeeping for software and other technology systems. Much like in your own home, if you avoid doing the housekeeping for software—improving performance, increasing test coverage, and so on—it leads to a great big mess.

technology stack: The collection of software used to build and deliver an application or service. Programmers often refer to “tech stacks” by acronyms based on the stack components. For instance, the LAMP stack is for Linux, Apache, MySQL, and Perl/PHP/Python; ELK stack is Elasticsearch, Logstash, and Kibana; and JAMstack is JavaScript, APIs, and Markup.

thought leadership: A buzzword referring to content created by someone who is (or wishes to be seen as) an expert in a field. The creator intends the content to be insightful and innovative, setting the direction for continuing conversations on the topic. Many in technical fields use the term in a mocking way and are especially dismissive of those who apply the term *thought leader* to themselves.

throw over the wall: A style of project release where an entity releases a project under a FOSS license and has no intention or plan for engaging with it afterward. The project originator will not provide maintenance, community management, responses to bug reports, or anything else for the project beyond releasing it.

TL;DR: Common internet acronym for “too long; didn’t read.” Used either to introduce a sort of executive summary of a long work of writing or in reply to someone else’s work notifying them (often sarcastically) that the respondent didn’t read the work because, really, who has the time to wade through all those words?

trade dress: The unique visual elements and characteristics of a product and its packaging. Trade dress is protected under intellectual property law, though the method and degree of protection varies by jurisdiction.

trademark: Distinguish the goods and services of one company or party from those of others. Trademarks include things like product names, logos, and taglines. They are protected under intellectual property law.

transformative work: A type of fair use where the author has taken a copyrighted work and changed it so that it has a new meaning or purpose. See also *fair use*.

upstream: The default mode of collaboration in FOSS projects and communities, where contributions are sent back to the project itself rather than keeping them for internal use. See also *downstream*.

value-add: Product features—usually available at an increased price—that address more customer needs, adding more value to the overall product offering. Value-add is an important element of the freemium and open core styles of business model.

vendoring: The practice of maintaining and using copies of the third-party software used by a project or organization. The software build uses these copies of the dependencies rather than going out and retrieving the latest versions from the internet (also known as *upstream versions* of the component).

viral license: A pejorative term for *reciprocal license*.

white-labeling: Acquiring a product or service from elsewhere, rebranding it with your organization's trade dress, then selling it as your own product.

work: Something you put effort into creating. Writing, music composition, and construction are examples of ways to create a work. Works often receive protection under intellectual property law.

work for hire: A work created on behalf of another entity, usually by way of some sort of employment or contractual engagement. The employer typically retains any intellectual property (IP) created in a work for hire (aka “work made for hire”) situation. This IP ownership can (and should) be defined in the contract or employment agreement.

YAGNI: Acronym for “you ain’t gonna need it.” See also *premature optimization*.

YOLO: Acronym for “you only live once.” Internet slang used when doing (or encouraging someone else to do) something that might not be especially wise but that you’re going to do anyway. Sometimes used as a verb, “We’re just gonna YOLO it in this product release, and if anything breaks, we’ll fix it later.”

Links from Chapters

Preface

- *Forge Your Future with Open Source*: <https://fossforge.com>
- FOSS Books book recommendations: <https://fossbooks.com>

Part I: Fundamentals of FOSS

Chapter 1: Lay the Foundation

- Why Home Depot Struggles and IKEA Thrives in China?: <https://www.forbes.com/sites/helenwang/2011/02/10/why-home-depot-struggles-and-ikea-thrives-in-china/?sh=5645ccfb18e4>
- *Commission on New Technological Uses of Copyrighted Works (CONTU)* on Wikipedia: <https://en.wikipedia.org/wiki/CONTU>
- ih cross-stitch pattern creator on GitHub: <https://github.com/glasnt/ih>
- The State of Enterprise Open Source—A Red Hat Report (2022): <https://www.redhat.com/en/resources/state-of-enterprise-open-source-report-2022>

Chapter 2: Make the Intangible Profitable with Intellectual Property

- World Intellectual Property Organization (WIPO): <https://www.wipo.int/portal/en/index.html>
- Why so little love for the patent grant in the MIT License?: <https://opensource.com/article/18/3/patent-grant-mit-license>
- Understanding Copyright and Related Rights: https://www.wipo.int/edocs/pubdocs/en/wipo_pub_909_2016.pdf
- Copyright in General: <https://www.copyright.gov/help/faq/faq-general.html>
- Copyright Protection of Computer Software: <https://www.wipo.int/copyright/en/activities/software.html>
- What Is Not Protected by Copyright: <https://www.copyright.gov/circs/circ31.pdf>

- Originality Requirements under U.S. and E.U. Copyright Law: <https://softwarefreedom.org/resources/2007/originality-requirements.html>
- Berne Convention for the Protection of Literary and Artistic Works: <https://www.wipo.int/treaties/en/ip/berne/>
- pandas open source data analysis and manipulation tool: <https://pandas.pydata.org>
- Developer Certificate of Origin: <https://developercertificate.org/>

Chapter 3: Licenses: The Rules of IP Engagement

- *Spoon Theory* on Wikipedia: https://en.wikipedia.org/wiki/Spoon_theory
- Open Source Initiative (OSI) Approved Licenses: <https://opensource.org/licenses>
- OSI License Review Process: <https://opensource.org/licenses/review-process/>
- Unity Reference-Only License: <https://unity.com/legal/licenses/unity-reference-only-license>
- Creative Commons: <https://creativecommons.org/>
- The State of Enterprise Open Source—A Red Hat Report (2022): <https://www.redhat.com/en/resources/state-of-enterprise-open-source-report-2021>

Chapter 4: Avoid Common FOSS Business Risks

- Software Heritage: <https://www.softwareheritage.org>
- What Happened When I Leaked My Server Password on GitHub.com: <https://web.archive.org/web/20230607132522/https://craighays.com/what-happened-when-leaked-server-password-github/>
- OWASP Secure Coding Practices—Quick Reference Guide: <https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/>
- *Cyber Resilience Act (CRA)* on Wikipedia: https://en.wikipedia.org/wiki/Cyber_Resilience_Act
- Community Health Analytics in Open Source Software (CHAOSS): <https://chaoss.community/>

Chapter 5: Strengthen Your Business Through FOSS

- The State of Enterprise Open Source—A Red Hat Report (2022): <https://www.redhat.com/en/resources/state-of-enterprise-open-source-report-2022>
- If the Linux Foundation was a software company, it'd be the biggest in the world: https://www.theregister.com/2023/09/28/kubecon_shanghai/
- Yocto Project: <https://www.yoctoproject.org>
- Ditto Project: <https://eclipse.dev/ditto/>
- This Fixable Problem [retention] Costs U.S. Businesses \$1 Trillion: <https://www.gallup.com/workplace/247391/fixable-problem-costs-businesses-trillion.aspx>

- How Investing in DEI Helps Companies Become More Adaptable: <https://hbr.org/2023/05/how-investing-in-dei-helps-companies-become-more-adaptable>
- Why DEI Is Important—Nine Benefits of Strong DEI in the Workplace: <https://insightglobal.com/blog/why-dei-is-important/>
- Why Is Diversity and Inclusion Important? Diversity in the workplace statistics: <https://learning.linkedin.com/resources/learning-culture/diversity-workplace-statistics-dei-importance>
- Outreachy: <https://www.outreachy.org>
- Women in Linux: <https://www.womeninlinux.org>
- Vets Who Code: <https://vetswhocode.io>
- Lesbians Who Tech: <https://lesbianswhotech.org/>
- CODE2040: <https://www.code2040.org/>

Chapter 6: Introducing the Open Source Program Office

- OSPO Alliance: <https://ospo-alliance.org>
- The TODO Group: <https://todogroup.org>
- OSPO++: <https://ospoplusplus.org>
- *Forge Your Future with Open Source*: <https://fossforge.com>
- InnerSource Commons: <https://innersourcecommons.org>
- *Program Management for Open Source Projects*: <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>
- Open Source Survey (2017): <https://opensourcesurvey.org/2017/>

Part II: Inbound (Using) FOSS

Chapter 7: Recognize and Acknowledge Your Inbound FOSS

- Creative Commons: <https://creativecommons.org>

Chapter 8: Inbound FOSS in Your Corporate Strategy

- Learn Wardley Mapping: <https://learnwardleymapping.com>
- Open Source Security and Risk Analysis Report (2022): <https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html?intcmp=sig-blog-ossra22>
- The State of Enterprise Open Source—A Red Hat Report (2022): <https://www.redhat.com/en/enterprise-open-source-report/2022>
- Is Open Source Software More Secure than Proprietary Products?: <https://www.govtech.com/security/is-open-source-software-more-secure.html>

Chapter 9: Establish Effective Inbound Policies and Procedures

- *Cognitive Load* on Wikipedia: https://en.wikipedia.org/wiki/Cognitive_load
- Good Leadership Is About Communicating “Why”: <https://hbr.org/2020/05/good-leadership-is-about-communicating-why>
- *The Checklist Manifesto* by Atul Gawande: <http://atulgawande.com/book/the-checklist-manifesto/>

Chapter 10: Get Started with Your Inbound Policies

- One Year After Log4Shell, Most Firms Are Still Exposed to Attack: <https://www.darkreading.com/application-security/one-year-later-log4shell-exposed-attack>
- *Cassandra* on Wikipedia: <https://en.wikipedia.org/wiki/Cassandra>
- U.S. General Services Administration (GSA) Open Source Policy: <https://open.gsa.gov/oss-policy/>

Chapter 11: Know the Links in Your Software Supply Chain

- *Heartbleed* on Wikipedia: <https://en.wikipedia.org/wiki/Heartbleed>
- *Log4Shell* on Wikipedia: <https://en.wikipedia.org/wiki/Log4Shell>
- *npm: Compromised and disruptively edited packages* on Wikipedia: https://en.wikipedia.org/wiki/Npm#Compromised_and_disruptively-edited_packages
- U.S. Executive Order on Improving the Nation’s Cybersecurity (EO 14028): <https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/>
- Software Heritage: <https://www.softwareheritage.org>
- Vulnerability-Exploitability eXchange (VEX)—An Overview: https://www.ntia.gov/files/ntia/publications/vex_one-page_summary.pdf
- CVE Program: <https://www.cve.org>
- NIST National Vulnerability Database (NVD): <https://nvd.nist.gov/vuln>
- GitHub Advisory Database: <https://github.com/advisories>
- CISA Known Exploited Vulnerabilities Catalog: <https://www.cisa.gov/known-exploited-vulnerabilities-catalog>
- ClearlyDefined: <https://clearlydefined.io>
- U.S. National Cybersecurity Strategy (2023): <https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf>
- Package URL (purl) Project: <https://www.aboutcode.org/projects/package-url.html>
- Scancode Toolkit: <https://github.com/nexB/scancode-toolkit>
- Fossology: <https://www.fossology.org>
- OSS Review Toolkit (ORT): <https://oss-review-toolkit.org/>
- Tern Project: <https://github.com/tern-tools/tern>
- CISA on Software Bill of Materials (SBOM): <https://www.cisa.gov/sbom>

- NTIA on The Minimum Elements for a Software Bill of Materials (SBOM): https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
- SPDX Example: <https://github.com/spdx/spdx-examples/blob/master/example8/spdx/examplemaven-0.0.1.spdx.json>
- CycloneDX Example: <https://github.com/CycloneDX/bom-examples/blob/master/SBOM/cern-lhc-vdm-editor-e564943/bom.json>

Chapter 12: Maintain FOSS License Compliance

- Four Microsoft licensing missteps that could cost you big bucks: <https://web.archive.org/web/20221201124614/https://www.techrepublic.com/article/four-microsoft-licensing-missteps-that-could-cost-you-big-bucks/>
- Nutanix Objects Violates MinIO's Open Source License: <https://blog.min.io/nutanix-objects-violates-minios-open-source-license/>
- *BusyBox GPL Lawsuits* on Wikipedia: https://en.wikipedia.org/wiki/BusyBox#GPL_lawsuits
- Software Freedom Conservancy—Current Status of Vizio Case: <https://sfconservancy.org/copyleft-compliance/vizio.html>
- Apache License v2.0 and GPL Compatibility: <https://www.apache.org/licenses/GPL-compatibility.html>
- Third Party Notices for Microsoft Visual Studio Code: <https://github.com/microsoft/vscode/blob/main/ThirdPartyNotices.txt>
- The Principles of Community-Oriented GPL Enforcement: <https://sfconservancy.org/copyleft-compliance/principles.html>
- Tern: a software package inspection tool to create a Software Bill of Materials (SBOM) for containers: <https://github.com/tern-tools/tern>
- Open Source Initiative (OSI) Approved Licenses: <https://opensource.org/licenses/>
- OSS Review Toolkit (ORT): <https://oss-review-toolkit.org/>

Chapter 13: Sustain. Maintain. Retain.

- *Our Common Future (Brundtland Report)* on Wikipedia: https://en.wikipedia.org/wiki/Our_Common_Future
- UN Sustainable Development Goals: <https://www.un.org/sustainabledevelopment/>
- *Corporate Sustainability* on Wikipedia: https://en.wikipedia.org/wiki/Corporate_sustainability
- Corporate Sustainability at a Crossroads—Progress Toward Our Common Future in Uncertain Times: <https://sloanreview.mit.edu/projects/corporate-sustainability-at-a-crossroads/>
- Ecosystem Definition in Merriam-Webster: <https://www.merriam-webster.com/dictionary/ecosystem>
- *Thinking in Systems*: <https://www.chelseagreen.com/product/thinking-in-systems/>

- Roads and Bridges—The Unseen Labor Behind Our Digital Infrastructure: <https://www.fordfoundation.org/work/learning/research-reports/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/>
- Open Source is Bigger Than You Can Imagine: <https://anchore.com/blog/open-source-is-bigger-than-you-imagine/>
- *Hearbleed* on Wikipedia: <https://en.wikipedia.org/wiki/Hearbleed>
- *Log4Shell* on Wikipedia: <https://en.wikipedia.org/wiki/Log4Shell>
- CHAOSS—All Metrics: <https://chaoss.community/kbtopic/all-metrics/>

Part III: Outbound (Contributing to and Releasing) FOSS

Chapter 14: Level Up Your Business with Outbound FOSS

- *Program Management for Open Source Projects*: <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>
- IBM Plex font: <https://github.com/IBM/plex>
- New York Public Library (NYPL) Core Ontology: <https://github.com/NYPL/nypl-core>
- To Recruit and Retain the Best Tech Talent, Companies Need to Use Open Source Software: <https://builtin.com/recruiting/use-open-source-software>
- In the era of open hybrid cloud, open source skills matter more than proprietary software skills: <https://developer.ibm.com/blogs/oreilly-open-source-skill-survey-blog/>

Chapter 15: Outbound Policies and Procedures

- Open Source at GitLab: <https://about.gitlab.com/handbook/engineering/open-source/>
- GitHub Balanced Employee IP Agreement (BEIPA): <https://github.com/github/balanced-employee-ip-agreement>
- Microsoft Open Source Program Documentation: <https://opensource.microsoft.com/program/>
- Citi FOSS Contribution Guidelines: <https://github.com/Citi/citi-ospo/blob/main/FOSS%20Contribution%20Guidelines.md>
- Zalando's Open Source Documentation: <https://opensource.zalando.com/docs>
- Open Source Policy Examples and Templates: <https://github.com/todogroup/policies>
- The TODO Group: <https://todogroup.org>

Chapter 16: Strategic Benefits and Risks of Outbound FOSS

- CHAOSS Project: <https://chaoss.community>
- *Forge Your Future with Open Source*: <https://fossforge.com>

- Making the Business Case for Corporate Philanthropy: <https://corpgov.law.harvard.edu/2011/08/20/making-the-business-case-for-corporate-philanthropy/>
- We Just Gave \$500,000 to Open Source Maintainers: <https://blog.sentry.io/we-just-gave-500-000-dollars-to-open-source-maintainers/>
- Make Your Employer Brand Stand Out in the Talent Marketplace: <https://hbr.org/2022/02/make-your-employer-brand-stand-out-in-the-talent-marketplace>
- Impact of Open Source on Recruitment—Case of Digital Companies: <https://ibima.org/accepted-paper/impact-open-source-recruitment-case-digital-companies/>
- Eclipse Software Defined Vehicle Project: <https://sdv.eclipse.org>
- Netflix Open Source Software Center: <https://netflix.github.io>
- Simian Army Suite of Tools: <https://github.com/Netflix/SimianArmy>
- Chaos Monkey Resiliency Testing Tool: <https://github.com/netflix/chaosmonkey>
- FLOSS Foundations Directory: <https://flossfoundations.org/foundation-directory/>
- Font Awesome: <https://fontawesome.com>
- RISC V International: <https://riscv.org>
- Creative Commons Attribution 4.0 International License: <https://creativecommons.org/licenses/by/4.0/>
- RISC V ISA Manual: <https://github.com/riscv/riscv-isa-manual>
- Agile Manifesto: <http://agilemanifesto.org>
- *The European Union Cyber Resilience Act* on Wikipedia: https://en.wikipedia.org/wiki/Cyber_Resilience_Act
- *The European Union General Data Protection Regulation* on Wikipedia: https://en.wikipedia.org/wiki/General_Data_Protection_Regulation
- U.S. Executive Order on Improving the Nation's Cybersecurity (EO 14028): <https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/>
- *Indian Digital Personal Data Protection Act of 2023* on Wikipedia: https://en.wikipedia.org/wiki/Digital_Personal_Data_Protection_Act%2C_2023
- Indian National Cyber Security Strategy 2020: <https://database.cyberpolycyportal.org/api/files/1664377398590fe3kzyt8xwd.pdf>
- *Open Source Law, Policy and Practice (2nd edn)*, Chapter 12: Export Control: <https://academic.oup.com/book/44727/chapter/378967490>
- Understanding US export controls with open source projects: <https://www.linuxfoundation.org/resources/publications/understanding-us-export-controls-with-open-source-projects>
- NIST Secure Software Development Framework (SSDF): <https://csrc.nist.gov/Projects/ssdf>
- SAFECODE Secure Development Practices: <https://safecode.org/category/resource-secure-development-practices/>

- OpenSSF Best Practices Working Group: <https://github.com/ossf/wg-best-practices-os-developers>

Chapter 17: Master Contribution Fundamentals

- *Forge Your Future with Open Source*: <https://fossforge.com>
- Drupal Association: <https://www.drupal.org/association/board>
- GitHub Sponsors: <https://github.com/sponsors>
- OpenCollective: <https://opencollective.com>
- Software Freedom Conservancy: <https://sfconservancy.org>
- Software in the Public Interest: <https://www.spi-inc.org>
- *Investing in Open Source—FOSS Contributor Fund*: https://github.com/indeedeng/FOSS-Contributor-Fund/blob/main/Investing_in_Open_Source-FOSS_Contributor_Fund.pdf
- FOSS Funders: <https://fossfunders.com>
- *Linux Distribution* on Wikipedia: https://en.wikipedia.org/wiki/Linux_distribution
- Python Software Foundation trademark policy: <https://www.python.org/psf/trademarks/>
- Configuring git: <https://git-scm.com/docs/git-config>
- Setting Your Commit Email Address in GitHub: <https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address>
- GitLab Profile documentation: <https://docs.gitlab.com/ee/user/profile/>
- CHAOSS Metrics Knowledgebase: <https://chaoss.community/kb-metrics-and-metrics-models/>
- *Allie Brosh* on Wikipedia: https://en.wikipedia.org/wiki/Allie_Brosh
- CHAOSS Software: <https://chaoss.community/software/>
- Bitergia: <https://bitergia.com>

Chapter 18: Develop Your Contribution Strategies

- *Making Process Improvements Stick*: <https://hbr.org/2018/11/making-process-improvements-stick>
- Outreachy: <https://outreachy.org>
- *501(c)(3)* on Wikipedia: [https://en.wikipedia.org/wiki/501\(c\)_organization#501\(c\)\(3\)](https://en.wikipedia.org/wiki/501(c)_organization#501(c)(3))

Chapter 19: The Strategy of Releasing FOSS

- bptop Project from Netflix: <https://github.com/Netflix/bptop>
- Porsche Design System: <https://github.com/porsche-design-system/porsche-design-system>
- UK Government Digital Service Open Source Repositories: <https://github.com/alphagov>

- *The Picture of Dorian Gray* by Oscar Wilde: <https://www.gutenberg.org/cache/epub/174/pg174-images.html#chap01>
- The Eclipse Foundation Antitrust Policy: https://www.eclipse.org/org/documents/Eclipse_Antitrust_Policy.pdf
- Affero General Public License version 3.0: <https://www.gnu.org/licenses/agpl-3.0.en.html>
- Open Source Definition: <https://opensource.org/osd>
- Creative Commons Attribution 4.0 International License: <https://creativecommons.org/licenses/by/4.0/>
- *SMART Goals Criteria* on Wikipedia: https://en.wikipedia.org/wiki/SMART_criteria
- Locke and Latham Method: https://med.stanford.edu/content/dam/sm/s-spire/documents/PD.locke-and-latham-retrospective_Paper.pdf
- *Objectives and Key Results (OKRs)* on Wikipedia: https://en.wikipedia.org/wiki/Objectives_and_key_results
- Overview of Creative Commons Licenses: <https://creativecommons.org/share-your-work/cclicenses/>
- Open Data Commons: <https://opendatacommons.org>
- Open Source Hardware Association Best Practices for Sharing: <https://www.oshwa.org/sharing-best-practices/>
- Understanding US export controls with open source projects: <https://www.linuxfoundation.org/resources/publications/understanding-us-export-controls-with-open-source-projects>
- A Longitudinal Examination of Net Promoter and Firm Revenue Growth: <https://www.jstor.org/stable/30163980>
- Chaos Monkey GitHub Repository: <https://github.com/netflix/chaosmonkey>
- Gerrit Project: <https://opensource.google/projects/gerrit>
- SecHub GitHub Repository: <https://github.com/mercedes-benz/sechub>
- FLOSS Foundations Directory: <https://flossfoundations.org/foundation-directory/>
- The Linux Foundation: <https://www.linuxfoundation.org>
- The Open Hive Foundation: <https://openhive.foundation>
- Creative Commons Attribution-ShareAlike 4.0 International License: <https://creativecommons.org/licenses/by-sa/4.0/deed.en>
- CERN Open Hardware Licence Version 2—Strongly Reciprocal: https://ohwr.org/cern_ohl_s_v2.txt

Chapter 20: Releasing FOSS Projects

- *Producing Open Source Software—How to Run a Successful Free Software Project* by Karl Fogel: <https://producingoss.com>
- Creative Commons Attribution-ShareAlike 4.0 International License: <https://creativecommons.org/licenses/by-sa/4.0/>

- *Psychological Safety* on Wikipedia: https://en.wikipedia.org/wiki/Psychological_safety
- Kubernetes: <https://kubernetes.io>
- “Unforced error” definition: <https://www.merriam-webster.com/dictionary/unforced%20error>
- Netflix Simian Army: <https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116>
- Inclusive Naming Project: <https://inclusivenaming.org>
- American Psychological Association Inclusive Language Guide: <https://www.apa.org/about/apa/equity-diversity-inclusion/language-guidelines>
- The Responsible Communication Style Guide: <https://rcstyleguide.com>
- GitHub Search: <https://github.com/search>
- GitLab Explore: <https://gitlab.com/explore/projects>
- Apache Software Foundation Projects: <https://projects.apache.org/projects.html>
- Eclipse Foundation Projects: <https://projects.eclipse.org>
- Linux Foundation Projects: <https://www.linuxfoundation.org/projects>
- Manual page for the UNIX/Linux top command: <https://man7.org/linux/man-pages/man1/top.1.html>
- FOSS Governance Collection: <https://foss-governance.org>
- Supply-Chain Levels for Software Artifacts: <https://slsa.dev>
- *Fuzzing* on Wikipedia: <https://en.wikipedia.org/wiki/Fuzzing>
- GitHub Search Results for “damn”: <https://github.com/search?q=damn+&type=code>
- “Bad Words” List from Luis von Ahn’s Research Group: <https://www.cs.cmu.edu/~biglou/resources/bad-words.txt>
- *George Carlin’s “Seven Dirty Words”* on Wikipedia: https://en.wikipedia.org/wiki/Seven_dirty_words
- OW2 Market Readiness Levels for Open Source Projects: <https://www.ow2.org/view/MRL/>
- Understanding Open Source Governance Models: <https://www.redhat.com/en/blog/understanding-open-source-governance-models>
- *Program Management for Open Source Projects—How to Guide Your Community-Driven, Open Source Project* by Ben Cotton: <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>
- The 2019 Tidelift managed open source survey results: <https://tidelift.com/subscription/managed-open-source-survey>
- The 2017 Open Source Survey results: <https://opensource-survey.org/2017/>
- *Forge Your Future with Open Source—Build Your Skills. Build Your Network. Build the Future of Technology* by VM (Vicky) Brasseur: <https://fossforge.com>
- The Good Docs Project: <https://thegooddocsproject.dev>
- Annual Intel Open Source Community Survey Results: <https://www.intel.com/content/www/us/en/developer/articles/community/annual-intel-open-source-community-survey-results.html>
- *SCO-Linux Disputes* on Wikipedia: https://en.wikipedia.org/wiki/SCO-Linux_disputes

- CHAOSS Metrics and Metrics Models: <https://chaoss.community/kb-metrics-and-metrics-models/>
- CHAOSS Software: <https://chaoss.community/software/>
- GitHub: <https://github.com>
- GitLab: <https://gitlab.com>
- Codeberg: <https://codeberg.org>
- Forgejo: <http://forgejo.org>
- git Version Control System: <https://git-scm.com>
- FLOSS Foundations Directory: <https://flossfoundations.org/foundation-directory/>
- The Europa International Foundation Directory 2022: <https://www.routledge.com/The-Europa-International-Foundation-Directory-2022/Publications/p/book/9781032274300>
- Worldwide NGO Directory: <https://www.wango.org/resources.aspx?section=ngodir>
- Apereo Foundation: <https://www.apereo.org>
- NumFOCUS: <https://numfocus.org>
- OW2: <https://www.ow2.org>
- The Apache Software Foundation: <http://apache.org>
- The Eclipse Foundation: <http://eclipse.org>
- Software in the Public Interest: <https://www.spi-inc.org>
- GuideStar: <https://www.guidestar.org>
- ProPublica Nonprofit Explorer: <https://projects.propublica.org/nonprofits/>
- EPRS Ideas Paper, Towards a more resilient EU—Digital sovereignty for Europe: [https://www.europarl.europa.eu/RegData/etudes/BRIE/2020/651992/EPRS_BRI\(2020\)651992_EN.pdf](https://www.europarl.europa.eu/RegData/etudes/BRIE/2020/651992/EPRS_BRI(2020)651992_EN.pdf)
- Instructions to Create an SPDX SBOM with the REUSE Tools: <https://reuse.software/faq/#bill-of-materials>
- Software Freedom Conservancy: <https://sfconservancy.org2>
- SPDX License List: <https://spdx.org/licenses/>
- SPDX License List GitHub Repository: <http://github.com/spdx/license-list-XML/>
- REUSE: <https://reuse.software>
- SPDX—Handling License Info: <https://spdx.dev/learn/handling-license-info/>
- How to Use GNU Licenses for Your Own Software: <https://www.gnu.org/licenses/gpl-howto.en.html>
- Applying the Apache License, Version 2.0 to Software: <https://www.apache.org/legal/apply-license.html>
- Assembling LICENSE and NOTICE files: <https://infra.apache.org/licensing-howto.html>

Chapter 21: Engage with Your FOSS Projects

- *Program Management for Open Source Projects—How to Guide Your Community-Driven, Open Source Project* by Ben Cotton: <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>
- *Field of Dreams* on Wikipedia: https://en.wikipedia.org/wiki/Field_of_Dreams
- Outreachy: <https://www.outreachy.org>
- Google Summer of Code: <https://summerofcode.withgoogle.com>
- Google Season of Docs: <https://developers.google.com/season-of-docs>
- Linux Foundation Mentorship Programs: <https://www.linuxfoundation.org/about/mentorship-programs/>
- Saving GitHub Repositories with Stars: <https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars>
- Community Health Analytics in Open Source Software (CHAOSS): <https://chaoss.community>
- *Negativity Bias* on Wikipedia: https://en.wikipedia.org/wiki/Negativity_bias
- OpenSearch: <https://opensearch.org>
- Elastic Enterprise Search: <https://www.elastic.co/enterprise-search>
- OpenTofu: <https://opentofu.org>
- Terraform: <https://www.terraform.io>
- Valkey: <https://valkey.io>
- Redis: <https://redis.io>
- *Producing Open Source Software—How to Run a Successful Free Software Project* by Karl Fogel: <http://producingoss.com>
- *Sunk Cost Fallacy* on Wikipedia: https://en.wikipedia.org/wiki/Sunk_cost#Fallacy_effect
- *The Human Nature of Failure and Resiliency* by VM Brasseur: <https://archive.org/details/redeploy2018-failure>
- *Manage Your Project Portfolio—Increase Your Capacity and Finish More Projects* by Johanna Rothman: <https://www.jrothman.com/books/manage-your-project-portfolio-increase-your-capacity-and-finish-more-projects/>
- *Principle of Least Astonishment* on Wikipedia: https://en.wikipedia.org/wiki/Principle_of_least_astonishment
- Archiving Repositories on GitHub: <https://docs.github.com/en/repositories/archiving-a-github-repository/archiving-repositories>

Chapter 22: Incorporate FOSS into Business Models

- *Business Model Generation* by Osterwalder and Pigneur: <https://www.strategyzer.com/library/business-model-generation>
- Humanitarian OpenStreetMap Team (HOT): <https://www.hotosm.org>
- United Nations Sustainable Development Goals: <https://www.un.org/sustainabledevelopment/sustainable-development-goals/>

- *Business Model* on Wikipedia: https://en.wikipedia.org/wiki/Business_model#Examples
- Open Source Review Toolkit (ORT): <https://oss-review-toolkit.org/ort/>
- *Gnomes Episode of South Park* on Wikipedia: [https://en.wikipedia.org/wiki/Gnomes_\(South_Park\)](https://en.wikipedia.org/wiki/Gnomes_(South_Park))
- Open Source Definition: <https://opensource.org/osd>
- Creative Commons Licenses: <https://creativecommons.org/share-your-work/cclicenses/>
- Red Hat: <https://redhat.com>
- *Red Hat Linux* on Wikipedia: https://en.wikipedia.org/wiki/Red_Hat_Linux
- *Red Hat Enterprise Linux* on Wikipedia: https://en.wikipedia.org/wiki/Red_Hat_Enterprise_Linux
- GitLab: <https://gitlab.com>
- *DevOps* on Wikipedia: <https://en.wikipedia.org/wiki/DevOps>
- GitLab Editions: <https://about.gitlab.com/install/ce-or-ee/>
- GitLab Documentation: <https://docs.gitlab.com>
- GitLab Handbook: <https://handbook.gitlab.com>
- GitLab Handbook License: <https://gitlab.com/gitlab-com/content-sites/handbook/-/blob/main/LICENSE>
- Grokability: <https://snipeitapp.com/company>
- Snipe-IT Is Open Source: <https://snipeitapp.com/product/open-source>
- Snipe-IT Support Plans: <https://snipeitapp.com/support>
- *Table 7. Survival of private sector establishments by opening year* from the Bureau of Labor Statistics: https://www.bls.gov/bdm/us_age_naics_00_table7.txt
- *Why Start-ups Fail* by Tom Eisenmann: <https://hbr.org/2021/05/why-start-ups-fail>
- *Freemium* on Wikipedia: <https://en.wikipedia.org/wiki/Freemium>
- Elastic Licensing FAQ: <https://www.elastic.co/pricing/faq/licensing>
- OpenSearch: <https://opensearch.org>
- HashiCorp Adopts Business Source License: <https://www.hashicorp.com/blog/hashicorp-adopts-business-source-license>
- OpenTofu: <https://opentofu.org>
- Redis Adopts Dual Source-Available Licensing: <https://redis.com/blog/redis-adopts-dual-source-available-licensing/>
- Redict: <https://redict.io>
- Valkey: <https://github.com/valkey-io/valkey>

Worksheets and Resources

FOSS License Compatibility Matrix

...with a project with this one & what is the likely license of the end product?

	MIT/X11	Apache v2	3-clause/ New BSD	2-clause/ Free BSD	MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
MIT/X11		any	any	any	MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
Apache v2	any		any	any	MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
3-clause/ New BSD	any	any		any	MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
2-clause/ Free BSD	any	any	any		MPL v2.0	LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
MPL v2.0	MPL v2.0	MPL v2.0	MPL v2.0	MPL v2.0		LGPL v2.1	LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
LGPL v2.1	LGPL v2.1	LGPL v2.1	LGPL v2.1	LGPL v2.1	LGPL v2.1		LGPL v2.1+	LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
LGPL v2.1+	LGPL v2.1+	LGPL v2.1+	LGPL v2.1+	LGPL v2.1+	LGPL v2.1+	LGPL v2.1+		LGPL v3 or LGPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+	LGPL v3 or LGPL v3+		GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v3 or GPL v3+	GPL v2	GPL v2+	GPL v3 or GPL v3+	AGPL v3
AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	AGPL v3	

Can I combine a project with this license...

Outbound FOSS Contribution Planning


Budget


Time


Skills


Strategy


Leadership

Outbound FOSS Planning

Date:

CRITICAL PROJECTS

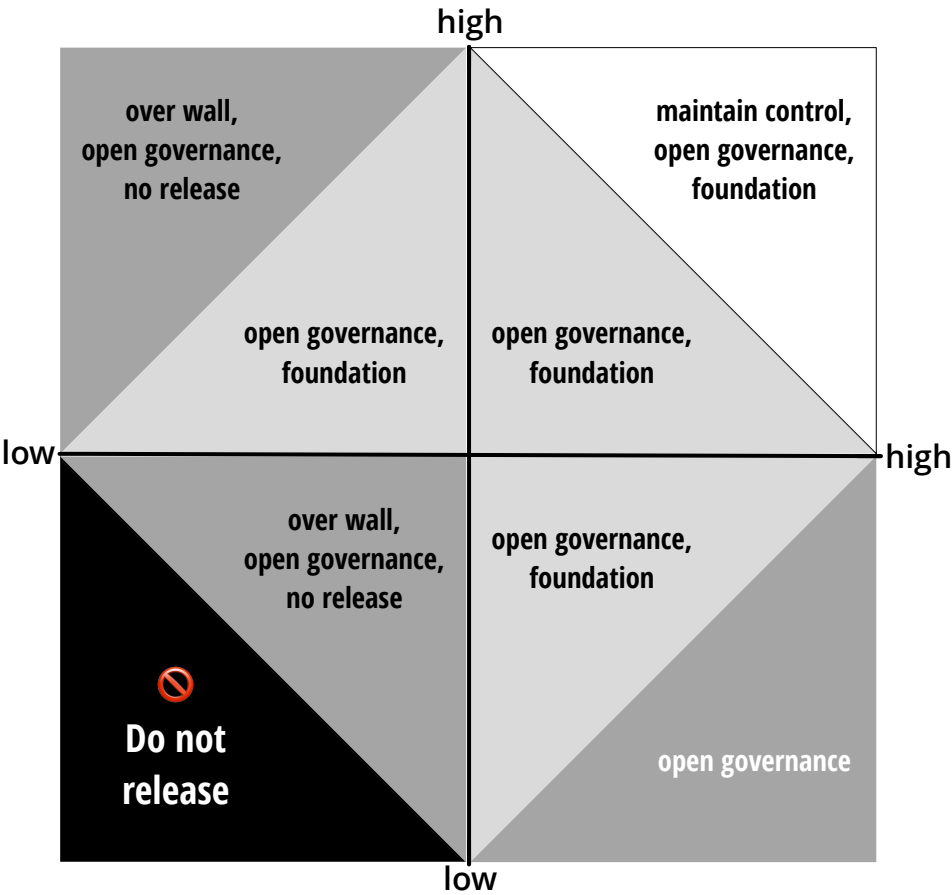
PROJECT NEEDS

CONTRIBUTION STRATEGY

P & P AND ONBOARDING

COMMUNICATION

Choose a Release Style



Other Useful Resources

This appendix provides pointers to other resources that can help you and your company be more successful with your free and open source software efforts. These are resources that I personally recommend to companies and individuals who want to learn how to do FOSS well.

Since the FOSS resource ecosystem is always evolving, you can also find these recommendations at <https://fossbooks.com>. That website will remain up-to-date with the latest recommended resources. It also gives you a handy reference in the (undoubtedly unlikely) event you don't have this book within reach.

Forge Your Future with Open Source

The only book about how to contribute to open source and Free Software projects and communities.

- Author: VM (Vicky) Brasseur
- Publisher: The Pragmatic Bookshelf
- Link: <https://fossforge.com>

Program Management for Open Source Projects

Ben skillfully leads you through the concepts necessary for the long-term success of any FOSS community and project.

- Author: Ben Cotton
- Publisher: The Pragmatic Bookshelf
- Link: <https://pragprog.com/titles/bcosp/program-management-for-open-source-projects/>

Producing Open Source Software

The original and still the best and most highly detailed book about the nuts and bolts of operating a FOSS project. Available either in print or in an online open access version.

- Author: Karl Fogel
- Publisher: O'Reilly Media and self
- Print link (first edition): <http://www.oreilly.com/catalog/producingoss/>
- Open access online link (second edition): <https://producingoss.com>

The Good Governance Handbook

A community-created and -maintained handbook targeted at companies trying to establish and improve their FOSS initiatives.

- Author: The OSPO Alliance community
- Publisher: The OSPO Alliance
- Link: <https://ospo-alliance.org/ggi/>

Intellectual Property and Open Source

A detailed yet approachable work that dives into the legal complexities of FOSS.

- Author: Van Lindberg
- Publisher: O'Reilly Media
- Link: <https://www.oreilly.com/library/view/intellectual-property-and/9780596517960/>

Open Source Law, Policy and Practice (Second Edition)

This book covers a number of different FOSS topics, not all of which are purely legal in nature. It's a good place to start if you need an introduction to a more complex issue, such as export control. Available either in print or in an online open access version.

- Editor: Amanda Brock
- Publisher: Oxford University Press
- Print link: <https://global.oup.com/academic/product/open-source-law-policy-and-practice-9780198862345?cc=us&lang=en&#>
- Open access online link: <https://academic.oup.com/book/44727>

The Business Value of Developer Relations

While not specifically about free and open source software, the community focus of Mary's book makes it a must-have for companies looking to release and maintain their own FOSS projects.

- Author: Mary Thengvall
- Publisher: Apress
- Link: <https://link.springer.com/book/10.1007/978-1-4842-3748-9>

Business Model Generation

A good introduction to and guide for creating a business model. Includes and is based around the popular *Business Model Canvas* framework.

- Authors: Alexander Osterwalder and Yves Pigneur
- Publisher: John Wiley & Sons, Inc.
- Link: <https://www.strategyzer.com/library/business-model-generation>

Index

DIGITS

3-Clause BSD License,
see BSD-3-Clause License

A

Advisory Database, [187](#)
Affero General Public License,
see AGPL
aggregation, *mere*, [59](#)
agile development, [253–255](#)
Agile Manifesto, [253–255](#)
AGPL
 distribution and, [61](#)
 license compatibility and,
 [64](#)
 as network reciprocal li-
 cense, [48](#)
 popularity of, [44](#)
 SaaS and, [48](#), [61](#), [300](#),
 [346](#)
 Snipe-IT, [380](#)
 as strong reciprocal li-
 cense, [61](#)
 summary of, [53](#)
algorithms, [26](#), [59](#)
All Rights Reserved, [40](#)
Amazon, forking by, [45](#)
Amazon Web Services, [138](#)
American Psychological Asso-
 ciation Inclusive Language
 Guide, [326](#), [331](#)
analyze step, software compo-
 sition analysis, [186–189](#)
anticompetition policies, [300](#)
antitrust policies, [300](#)

Apache Software Foundation
 Contributor License
 Agreements, [36](#)
 license preferences, [346](#)
 name collisions, checking
 for, [326](#)
 project interests, [343](#)
 trademark, [20](#)
Apache Struts, [68](#)
Apache-2.0 License
 as business friendly, [347](#)
 complexity of, [40](#)
 disclaimer of warranty,
 [70](#)
 license compatibility and,
 [62](#)
 OpenSSL relicensing, [72](#)
 patent clause, [22](#), [49](#),
 [57](#), [346](#)
 as permissive license, [49](#)
 Rust and, [57](#)
 summary of, [53](#)
Apereo Foundation, [223](#), [343](#)
APIs, copyright and, [28](#)
approve lists, [165](#), [171](#), [265](#)
approved for internal use only
 lists, [166](#)
archiving projects, [366](#)
article placement, [357](#)
artifact stores, [192](#)
Asparouhova, Nadia, [217](#),
 [267](#)
attribution
 Creative Commons as-
 sets, [126](#)
 license compliance and,
 [206–208](#), [353](#)
 notices, [206](#)

authority and policies and
 procedures enforcement,
 [158](#)
automating
 license compliance en-
 forcement, [211](#)
 policies and procedures
 approvals, [165](#)
 policies and procedures
 enforcement, [156](#),
 [159](#), [211](#)
 software composition
 analysis, [181](#), [188](#), [191](#)

B

bare licenses, [41](#)
Bees4Less example
 contributions example,
 [289–295](#)
 contributions, tracking,
 [279–281](#)
 copyright, [26](#), [31](#)
 copyright statements,
 [350](#)
 cost structure and projec-
 tions, [375](#)
 digital transformation,
 [132–140](#)
 intentional acquisition,
 [176](#)
 naming projects and, [324](#)
 patents, [21](#)
 policies and procedures,
 [148–152](#), [158](#)
 releases, [308](#), [313–316](#)
 software supply chain,
 [129](#), [176](#)
 sustainability, [216](#), [246](#)

- Berne Convention for the Protection of Literary and Artistic Works, 29
 - Bitergia, 282
 - blockers, 190
 - Box, George, 134
 - bpftop, 297
 - brand reputation
 - community and benefits to, 90–92
 - community and risk to, 85
 - contribution benefits, 90–92, 263
 - FOSS advantages, 90–92
 - name collisions and brand dilution, 324
 - naming projects, 324–325
 - outbound FOSS benefits, 247
 - outbound FOSS risks, 257, 261
 - release benefits, 298
 - release risks, 309, 315
 - Bressers, Josh, 217
 - Brock, Amanda, 424
 - Brosh, Allie, 282
 - Brundtland Report, 213–215
 - BSD-3-Clause License
 - brevity of, 40
 - license compatibility and, 64
 - NumPy and, 125
 - summary of, 54
 - budget constraints
 - contributions, 283
 - release approaches, 320
 - build-versus-buy question, 140
 - burnout
 - dwindling maintenance and, 363
 - OSPO and, 109, 113
 - sustainability and, 218
 - Business Model Generation*, 425
 - business models, *see also* open core
 - vs. business plans, 370
 - charging for FOSS, 371, 373, 375
 - components of, 371
 - defined, 369
 - FOSS as element of, 372
 - integrating FOSS, 374–381
 - protective elements, 380
 - resources on, 425
 - success stories, 377–379
 - types of, 371
 - Business Source License (BUSL-1.1), 56
 - business strategy
 - constraints on contributions, 284
 - FOSS advantages, 98
 - goal frameworks, 302
 - inbound FOSS and, 131–144
 - license selection and, 44, 50
 - OSPO functions and, 103–107
 - outbound FOSS and, 131, 243–261
 - release decisions, 303–305, 312
 - release project types, 306–308
 - The Business Value of Developer Relations*, 425
 - BUSL-1.1 License, 56
 - BusyBox, 205
- ## C
-
- C/C++ library, 47
 - CAL-1.0 License, 46
 - CCLA (corporate CLA), 36, 275
 - CERN, 55, 314
 - CERN Open Hardware License, 55, 314
 - change and agile development, 255
 - Chaos Army, 250
 - chaos engineering, 250, 325
 - CHAOSS Project, 74, 221, 245, 281–282, 340, 360
 - The Checklist Manifesto*, 160
 - checklists, policies and procedures enforcement, 159, 257
 - Chief Open Source Officer, 114, 118
 - CI/CD (continuous integration/continuous deployment)
 - automated software composition analysis, 181
 - license enforcement, automating, 212
 - CISA (Cybersecurity and Infrastructure Security Agency), 187, 194
 - Citi, 240
 - CLAs, *see* Contributor License Agreements
 - ClearlyDefined, 187, 194
 - cloning, forking and, 45
 - closing projects, 366
 - cloud computing
 - development of, 88
 - digital transformation example, 134–140
 - reliability of, 138
 - storing credentials in, 260
 - Cloud Native Computing Foundation, 88
 - code, *see* source code
 - CODE2040, 95
 - Codeberg, 341
 - codes of conduct, 109, 221, 334, 360
 - cognitive load, 154
 - collaboration
 - agile development and, 254
 - FOSS stakeholders, 127
 - InnerSource, 104, 107, 152
 - OSPO functions, 104, 107
 - outbound FOSS benefits, 250
 - policies and procedures, 154
 - release benefits, 299
 - commercial open source software (COSS), *see* open core
 - commodity policies, 210, 212
 - common law and trademarks, 20
 - communication
 - deploying policies and procedures, 155
 - diversity and, 95
 - FOSS culture and, 5
 - jargon and, xxiii
 - obfuscation, avoiding, xxiv
 - promoting contributions, 288, 292
 - community
 - brand reputation benefits, 90–92

- brand reputation risks, 85
- building, 85
- codes of conduct, 109
- community-focused OSPOs, 109, 118
- Contributor License Agreements and, 34
- defined, 13
- diversity and, 96, 109
- enabling project evolution, 363
- forking and, 45, 362, 383
- importance of, 4, 14
- integrating FOSS with business model and, 381
- license compliance enforcement and, 203–204
- management, 85, 116, 360
- open core and, 382–384
- OSPO and, 104, 107–113, 116, 361
- patents and, 23, 276
- project health evaluation and, 221
- project names and, 81
- recruiting from, 92, 109, 247
- release approaches and, 310–313, 320
- releases and community management, 360
- sustainability policies, 169
- total cost of ownership and, 89
- toxic, 74, 85, 360
- trademarks and, 20
- withdrawing from project and, 365
- Community Health Analytics in Open Source Software Project, *see* CHAOSS Project
- community managers and OSPO, 116, 361
- competition
 - anticompetition policies, 300
 - FOSS advantages, 88, 98, 251
 - open core and, 382
 - outbound FOSS risks, 258
 - release benefits, 299
 - release risks, 76, 258
 - sustainability and, 219
- conferences
 - OSPOs and, 110, 112
 - releases and, 357
- configuration documentation, 334
- consensuality, 124
- Conservancy, *see* Software Freedom Conservancy
- Consul, 103
- consumption, *see* inbound FOSS
- containers
 - defined, 184
 - development of, 88
 - identifying inbound FOSS, 126
 - images and files, 69
 - licenses and, 184, 209
 - registries, 209
 - security and, 69
 - software composition analysis, 184
- content management systems in digital transformation example, 133–140, 150
- continuous integration/continuous deployment (CI/CD)
 - automated software composition analysis, 181
 - license enforcement, automating, 212
- contracts, licenses as, 41
- contributions
 - approve lists, 265
 - benefits of, 88, 246–249, 263
 - brand reputation benefits, 90–92, 263
 - committing to, 265
 - constraints, identifying, 283–285, 290
 - copyright and, 30–33, 274, 278
 - defining contribution strategy, 286, 291
 - downstream, 271–273
 - financial support, 224, 266–269
 - foundation memberships and support, 223
 - identifying critical projects, 285, 290
 - identifying project needs, 285, 291
 - importance of, 3, 14
 - IP considerations, 273–277
 - maintenance costs, sharing, 141–142
 - metrics, 281–282
 - naming projects and, 324
 - non-code, 266
 - onboarding contributors, 287–288, 292
 - OSPO and, 109, 265–266
 - without OSPO, 294
 - patents and, 275
 - planning worksheet, 293, 421
 - policies, 169, 171, 235–237, 273–274, 277, 287, 292
 - procedures, 241, 287, 292
 - project health and number of contributors, 221
 - quality review, 278
 - releases and contribution guidelines, 334
 - relicensing and, 72
 - review of, 97, 257, 278
 - risk and, 30–33, 255–261
 - security and, 97, 278
 - strategies example, 289–295
 - strategies for, 283–295
 - sustainability and, 169, 222, 246, 263
 - talent as, 265
 - time as, 264
 - tracking, 278–281
 - trademarks and, 276
 - types of, 14, 222, 264–269
 - upstream, 171, 270–273
- Contributor License Agreements
 - administrative overhead, 337
 - copyright transfers, 34
 - corporate CLA (CCLA), 36, 275
 - defined, 34
 - dislike of, 337
 - individual CLA (ICLA), 36
 - policies, 37, 236, 274
 - procedures, 241, 274
 - release preparation, 336, 339
 - relicensing and, 383
 - supporting contributions and, 265

- cooperation
 - defined, 98
 - OSPO and, 104
 - outbound FOSS benefits, 251
 - release benefits, 299
 - coordination, OSPO functions, 104
 - COPYING file, 349
 - copyleft licenses, *see* reciprocal licenses
 - copyright, *see also* Contributor License Agreements; Developer Certificates of Origin; IP; licenses
 - algorithms, 26
 - All Rights Reserved, 40
 - APIs, 28
 - assignment agreements, 29
 - automatic, 29, 39–40
 - basics of, 24–34
 - company vs. employee ownership, 29–30, 236
 - contributions and, 30–33, 274, 278
 - copyright statements, 325, 350–353
 - development of free software and, 6
 - employee agreements and, 30, 32, 234
 - expressed, original, and fixed criteria, 24, 27
 - infringement as federal issue, 42
 - jurisdiction and, 29, 33
 - limits on what can be copyrighted, 25
 - nontriviality and, 27
 - Oracle/Google suit, 28
 - ownership matrix, 30, 236
 - public domain and, 33, 54
 - reciprocal licenses and, 7
 - registering, 29
 - release preparation, 336–339
 - relicensing and, 72
 - shared authorship and, 27–33
 - software supply chain risks, 67–68
 - source code, 26
 - trolls, 203
 - work for hire and, 29
 - copyright statements, 325, 350–353
 - core capabilities in business models, 371
 - corporate CLA (CCLA), 36, 275
 - corporate authorship, 29
 - corporate espionage, 75
 - COSS, *see* open core
 - costs
 - in business models, 371, 375
 - lowering expenditures in digital transformation example, 136–140
 - maintenance, 83, 140, 142, 261
 - Not Invented Here syndrome and, 140
 - outbound FOSS, 245
 - staff departures, 93
 - total cost of ownership, 89
 - vendorship, 170
 - Cotton, Ben, 115, 335, 355, 361, 363, 423
 - crayon licenses, 51
 - Creative Commons Attribution 4.0 International License, 252
 - Creative Commons BY-SA 4.0 International License, 377
 - Creative Commons licenses
 - GitHub employee agreement, 237
 - identifying inbound FOSS, 126
 - non-software releases, 303
 - Producing Open Source Software*, 318
 - public domain and, 33, 54
 - RISC-V, 252
 - selling and, 377
 - types and options, 54
 - Wardley Maps, 134
 - Creative Commons Zero (CC0) License
 - GitHub employee agreement, 237
 - public domain and, 33, 54
 - creative works
 - copyright and license statements, 351
 - Creative Commons licenses and, 54, 303
 - outbound FOSS, 231
 - releases, 303, 308
 - selling, 377
 - Cryptographic Autonomy License, 46
 - cryptographic keys, reciprocal licenses, 46
 - culture
 - cultural appropriation, 326
 - FOSS as, 5, 12
 - CVE Program, 187
 - Cyber Resilience Act, 79
 - cybersecurity
 - EO 14028, 179, 192, 194, 258
 - National Cybersecurity Strategy, 190
 - Cybersecurity and Infrastructure Security Agency (CISA), 187, 194
 - CycloneDX, 195, 197
- ## D
-
- data
 - data rights and Cryptographic Autonomy License, 46
 - exposure risks, 77–79
 - licenses, 55
 - data centers, in digital transformation example, 133
 - dates in copyright and license statements, 352
 - DCOs, *see* Developer Certificates of Origin
 - Debian Free Software Guidelines, 9
 - dependencies
 - containers and, 184
 - inbound FOSS and, 125
 - policy and procedures, 157
 - release risks, 78, 82
 - scanning in software composition analysis, 182–186
 - derivatives
 - Creative Commons licenses, 54
 - defined, 58
 - distribution and, 60

- license compliance and providing source code, 208
 - licenses and, 46, 58–60, 345
 - reciprocal licenses and, 46–47, 59
 - designers, as FOSS stakeholders, 128
 - developer advocacy team, *see* developer relations team
 - Developer Certificates of Origin
 - defined, 35
 - policies, 37, 236, 274
 - procedures, 241, 274
 - release preparation, 336–339
 - developer evangelist, *see* developer relations team
 - developer relations team
 - OSPOs and, 110–111, 117
 - release promotion, 358
 - developers
 - as FOSS stakeholders, 128, 229
 - as IP policy stakeholders, 164
 - in OSPO, 116
 - policies and procedures input, 128
 - software composition analysis remediation, 190
 - development
 - agile development, 253–255
 - development-centric OSPOs, 111, 118
 - downstream, 269–273
 - OSPO and, 106, 111, 118
 - upstream, 269–273
 - digital sovereignty movement, 344
 - digital transformation
 - defined, 131
 - FOSS advantages, xviii, 87–89
 - inbound FOSS and corporate strategy, 131–140
 - time needed for, 139
 - disapproved at all times without VP signoff lists, 166
 - disaster planning, 168
 - disclaimer of warranty, 70, 79, 178
 - discoverability, 357
 - distinctiveness and trademarks, 20
 - distribution
 - defined, 60
 - derivatives and, 60
 - license compliance and, 204
 - licenses and, 48, 60–61, 204
 - network reciprocal licenses, 48
 - over a network, 61
 - Ditto, 93
 - diversity
 - FOSS advantages, 94–96
 - OSPO and, 109
 - security of code and, 97
 - documentation, *see also* policies; procedures
 - agile development and, 254
 - automated software composition analysis, 181
 - branding and, 325
 - closed projects, 366
 - governance, 310
 - importance of, 116, 333
 - onboarding contributors, 288, 292
 - outbound FOSS and documentation team, 230
 - outbound FOSS goals, 243
 - releases, 333–336
 - resources on, 335
 - software composition analysis, 188
 - technical writers and, 335
 - throwing it over the wall releases and, 335
 - domain names, 356
 - downstream
 - contributions, 271–273
 - development, 269–273
- ## E
- Eclipse Foundation, 326, 343, 346
 - Eclipse Public License, 48, 53
 - Eclipse Software Defined Vehicle, 249
 - ecosystems
 - building, 253
 - defined, 215
 - ecosystem ownership, 252, 300
 - forges and ecosystem resiliency, 342
 - release approaches and, 311
 - software supply chain as, 217
 - sustainability and, 215–217
 - education, OSPO and, 105
 - Eisenmann, Tom, 379
 - Elastic, 72, 362, 384
 - Elasticsearch, 45, 72, 362
 - employee agreements
 - copyright and, 30, 32
 - GitHub, 237
 - outbound FOSS and, 234
 - employees, *see also* onboarding; recruitment; retention
 - copyright and, 29–30, 236
 - diversity, 94–96
 - FOSS advantages, 92–96
 - hiring maintainers, 222
 - internships, 95, 358
 - professional development, 94, 232, 248
 - wages, 232
 - encryption and export controls, 259, 305
 - End User License Agreements (EULAs), 55
 - EO 14028, 179, 192, 194, 259
 - EPL, 48, 53
 - Equifax, 68
 - escalation path, policies and procedures, 158
 - espionage, corporate, 75
 - EULAs (End User License Agreements), 55
 - Europa International Foundation Directory, 342
 - European Union Cyber Resilience Act, 259
 - European Union General Data Protection Regulation, 259
 - events and messaging about releases, 357
 - exclusive licenses, 40

- Executive Order on Improving the Nation's Cybersecurity, *see* EO 14028
- export controls, 259, 305
- expressed criteria for copyright, 24
- ## F
-
- F/LOSS term, 6
- fair use
- defined, 42
 - Oracle/Google suit, 28
- Fastly, 91
- fauxpen source, *see also* open core
- licenses, 56, 208
 - relicensing and, 72
 - standards and, 16
- feedback
- agile development and, 254
 - outbound FOSS and, 254
 - release team and, 319
- files, containers, 69
- fiscal sponsorship, 267
- fixed criteria for copyright, 24
- FLOSS Foundations Directory, 342
- Fogel, Karl, 317, 335, 424
- Font Awesome, 251
- Forge Your Future with Open Source*, xix, 14, 107, 246, 335, 423
- Forgejo, 341
- forges
- adding licenses to, 349
 - defined, 117, 341
 - preparing forge for releases, 354
 - release mechanism, 354
 - selecting, 298, 340–344
 - self-hosting, 341
- forking
- cloning and, 45
 - community and, 45, 362, 383
 - defined, 45
 - downstream contributions, 271–273
 - internal forks, 271–273
 - policies, 171
 - relicensing and, 383
 - sustainability and, 362
- forward porting, 272
- FOSS, *see* free and open source software (FOSS)
- FOSS Contributor Funds, 267
- FOSS Funders, 268
- FOSS Governance Collection, 328, 333
- FOSS Open Stack project, 138
- FOSSology, 193
- foundations
- areas of interest, 343
 - corporate memberships and support, 223, 344
 - criteria for accepting projects, 344
 - donating projects to, 311, 320, 342, 346
 - lists of, 342
- free and open source software (FOSS), *see also* contributions; licenses; non-software; open core; releases
- allowing time for, xxii
 - benefits overview, 87–99
 - as business model element, 372
 - business model integration, 374–381
 - charging for, 301, 371, 373, 375
 - complexity of, xviii
 - as culture, 5, 12
 - development of, 6–11
 - Four Freedoms, 7, 12, 44
 - free term, 8–9
 - free vs. open source software, 11
 - innovation and, 87–89, 141
 - percentage of code as, 142
 - resources on, xxv, 107, 423–425
 - size of ecosystem, 5
 - software acquisition types, 176
 - ubiquity of, xvii, 3, 61, 66
- Free Software, Four Freedoms, 7
- Free Software
- charging for, 7
 - defined, 5
 - development of, 6–9
 - free term, 5
- free software
- charging for, 301
 - Four Freedoms, 12
 - free term, 8–9
 - vs. open source software, 11
- Free Software Foundation, 7, 44, 62
- Free Software Foundation Europe, 351
- FreeBSD Foundation, 224
- freemium, *see* open core
- fuzzing, 330
- ## G
-
- Gawande, Atul, 160
- General Services Administration, 171
- Getting Started Guides, 288, 292
- giggle test, 328
- Git
- Developer Certificates of Origin, 36, 338
 - forges, 341–342
 - identity management and, 280
 - notices, 207
 - ubiquity of, 220
- GitHub
- Advisory Database, 187
 - archiving projects, 366
 - Balanced Employee IP Agreement, 237
 - cloning and forking term, 45
 - identity management and, 280
 - name collisions, checking for, 326
 - popularity of, 298, 341
 - Stars, 359
- GitHub Sponsors, 267
- GitLab
- identity management and, 280
 - name collisions, checking for, 326
 - as open core, 382
 - Open Source policy, 237
 - popularity of, 341
 - success of, 378, 381
- GNU Affero General Public License, *see* AGPL
- GNU General Public License version 2, *see* GPLv2 License

- GNU General Public License
version 3, *see* GPLv3 License
- GNU Lesser Public License,
see LGPL
- GNU Manifesto, 7
- GNU Project, 6, 47
- GNU Public licenses, *see* GPL
- goal frameworks, 302
- The Good Governance Handbook*, 424
- Google, 28, 307, 358
- Google Season of Docs, 358
- Google Summer of Code, 358
- governance
documentation, 310
release preparation and, 332
releases with open governance, 310–313, 320
- GPL
COPYING file, 349
Vizio and right to repair, 205
- GPLv2 License
license compatibility and, 62
popularity of, 44
- GPLv3 License
complexity of, 40
license compatibility and, 62
popularity of, 44
as strong reciprocal license, 46
summary of, 53
termination clause, 204
- Grokability, 379–380
- Guidestar, 343
- ## H
- hardware
BOM formats for, 197
in digital transformation example, 133, 137
distribution of software and, 61
licenses, 55
release example, 314
- Hays, Craig, 77, 260
- head of OSPO, 114
- Heartbleed vulnerability, 178, 218
- high risk profile, 165
- The Home Depot, 4
- The Home Way, 4
- Hudson, 45
- Humanitarian Open-StreetMap Team (HOT), 370
- ## I
- IANAL, 18
- IBM, 8, 231
- ICLA (individual CLA), 36
- identity management and tracking contributions, 279–281
- ih, 13
- images, containers, 69
- inbound FOSS
awareness of, 123–130
corporate strategy and, 131–144
defined, 123
digital transformation and, 131–140
identifying, 125
Not Invented Here syndrome, avoiding, 140–142
policies, 140, 163–171, 209–212, 233–241
risks, 65–75, 143
security advantages, 143
stakeholders, 127
types of, 123
- Inbound Equals Outbound assumption, 339
- Inclusive Naming Initiative, 326, 331
- India Digital Personal Data Protection Act, 2023, 259
- India National Cyber Security Strategy 2020, 259
- individual CLA (ICLA), 36
- information security, 260
- InnerSource, 104, 107, 152
- innovation, FOSS benefits, 87–89, 141
- installation documentation, 334
- integration policies (vendor-ing), 169–171
- intellectual property, *see* IP
- Intellectual Property and Open Source*, 424
- intentionality, 124
- internal forks, 271–273
- internships, 95, 358
- Investing in Open Source*, 268
- IP, *see also* Contributor License Agreements; copyright; Developer Certificates of Origin; licenses; patents; trademarks
contributions and, 273–277
defined, 19
employee agreements and, 234
as foundation of FOSS, 17
importance of understanding, 18
policies, 163–166, 233–240, 256, 274
release risks, 75–86, 256
- issue trackers, 189
- IT, *see also* security
as FOSS stakeholder, 127
Not Invented Here syndrome and, 140
project website and blog, 357
release team and, 318
shadow IT, 177
software composition analysis, 180
- ## J
- jargon, xxiii, 147, 151
- Jenkins, 45
- ## K
- Kibana, 72, 384
- Kubernetes, 88, 322
- ## L
- language
inclusive, 331
jargon, xxiii, 147, 151
policies and procedures, 151
release preparation and, 331
- leadership
constraints on contributions, 285
release decisions and leadership support, 304
release team leadership, 319
- legal team
approve lists, 165

- as FOSS stakeholder, 127, 230
 - importance of, 18, 127
 - IP policy and, 164–165
 - license selection and, 344
 - stop lists, 166
 - Lesbians Who Tech, 95
 - Lesser GNU General Public License, *see* LGPL
 - LGPL
 - license compatibility and, 64
 - popularity of, 44
 - summary of, 53
 - as weak reciprocal license, 48
 - liability
 - avoiding through remediation, 190
 - Cyber Resilience Act and, 79
 - software supply chain and, 70
 - librarians, 152
 - libraries
 - identifying inbound FOSS, 125
 - reciprocal licenses and, 46
 - license compliance
 - attribution and, 206–208
 - containers and, 209
 - copyright trolls and profiteers, 203
 - costs of violations, 39
 - defined, 201
 - distribution and, 204
 - enforcement, 202–204
 - enforcement, automating, 211
 - infringement, consequences, 204
 - infringement, defined, 202
 - license compatibility and, 206
 - OSPO and, 105, 107, 118
 - policies, 209–212
 - principles of community-oriented enforcement, 203–204
 - releases and, 82, 353
 - risks, 67, 82, 211
 - software supply chain inspection, 201
 - software supply chain risks, 67, 82
 - software supply chain verification, 208
 - source code, providing, 208, 211
 - strategies, 206–209
 - LICENSE, LICENSE.txt, and LICENSE.md files, 349
 - licenses, *see also* Contributor License Agreements; Developer Certificate of Origin; fauxpen source; Open Source Definition (OSD); Open Source Initiative (OSI); permissive licenses; proprietary licenses; reciprocal licenses, IP; permissive licenses; reciprocal licenses
 - adding to forge, 349
 - agreement and validity, 40
 - applying, 348–353
 - approve lists for, 165, 171
 - bare, 41
 - charging for FOSS and, 301, 376
 - compatibility of, 61–64, 165, 206, 345
 - compatibility matrix, 62, 420
 - containers, 184, 209
 - as contracts, 41
 - crayon, 51
 - defined, 39
 - derivatives and, 46, 58–60, 345
 - development of, 7
 - distribution and, 48, 60–61, 204
 - donating projects to foundations and, 346
 - exclusive, 40
 - fauxpen source, 56, 208
 - FOSS advantages, 43–44
 - hardware, 55
 - integrating FOSS with business model and, 380
 - license identifiers, 351
 - license statements, 350–353
 - lists of, 52, 349
 - multi-licensing, 57, 348
 - non-open source, 54–57
 - OSD criteria, 9, 50
 - OSI certification process, 50–52
 - patent clauses, 22, 49, 57, 275, 346
 - policies, 52, 209–212
 - projects released without, 67, 82
 - releases, 82, 344–353
 - relinquishing copyright in, 33
 - selecting, 44, 50, 344–348
 - standardization of, 43, 52
 - stop lists for, 166
 - table of common, 53
 - types of, 44–50
 - verifying open source, 208
 - warranty clauses, 41, 70, 79, 178
 - Lindberg, Van, 424
 - linking, *see also* software supply chain
 - derivatives and, 59
 - reciprocal licenses and, 46
 - Linux
 - development of free software and, 8
 - Red Hat distributions, 377
 - SCO lawsuit, 339
 - Linux Foundation, 90, 311, 326
 - Linux Foundation Mentorship Programs, 358
 - Locke and Latham's Five Principles, 302
 - log4j, 167, 177–178, 218
 - Log4Shell vulnerability, 167, 177–178, 218
 - logs
 - IP policies, 163–166
 - security policies, 167
 - low risk profile, 165
- ## M
- Madrid, 103
 - maintainers
 - burnout, 218
 - financial support, 267
 - hiring, 222
 - reporting chain and, 223
 - sabotage by, 168, 179, 192
 - sponsored projects and, 224, 267
 - supporting, 222

- sustainability and, 217, 222
 - vs. vendors, 177
 - maintenance
 - containers, 69
 - corporate FOSS maintenance funds, 267
 - costs, 83, 140, 142, 261
 - dwindling, 363
 - FOSS advantages, 90, 249
 - internal forks and, 272
 - Not Invented Here syndrome and, 140
 - overdoing, 363
 - of policies and procedures, 153, 160, 165
 - release approaches and, 308–313
 - release decisions and, 303
 - resources on, 355
 - risks, 83
 - sustainability and, 74, 169
 - vendor, 170
 - Manage Your Project Portfolio*, 365
 - market research, outbound FOSS and, 230, 248
 - market trends
 - FOSS advantages, 88, 248
 - market ownership and outbound FOSS, 252
 - OSPO and, 106, 111
 - release benefits, 300
 - marketing
 - distrust of, 110
 - marketing team as FOSS stakeholder, 128
 - marketing-type OSPOs, 110, 118
 - metrics, 359
 - OSPOs and, 110, 115, 118
 - outbound FOSS and, 230
 - releases, 336, 354–359
 - technical marketers, 115, 356
 - Mercedes-Benz, 307
 - mere aggregation, 59
 - mergers and acquisitions
 - license compliance policies and, 211
 - naming releases and, 324
 - trademarks and, 81, 324
 - metadata
 - copyright and license statements, 351
 - software composition analysis, 185
 - metrics
 - contributions, 281–282
 - marketing, 359
 - project health, 74, 221
 - project value, 245
 - releases, 306, 340, 359
 - Microsoft
 - development of free software and, 9
 - license auditing, 205
 - notices, 207
 - outbound FOSS policy, 240
 - minIO, 205
 - MIT License
 - attribution in, 206
 - GitLab documents, 237, 378
 - implied patent grant, 23
 - license compatibility and, 64
 - as permissive license, 49
 - Rust and, 57
 - summary of, 54
 - text of, 11
 - variants of, 210
 - MITRE, 187
 - modding, 253
 - Mozilla Public License
 - license compatibility and, 64
 - popularity of, 44
 - summary of, 53
 - as weak reciprocal license, 48
- ## N
- names
 - company names, avoiding in project names, 323–325
 - discoverability and, 327
 - domain names, 356
 - giggle test, 328
 - inclusive, 325
 - name collisions, 324, 326
 - naming conventions, 322
 - project namespaces, 325
 - projects, 81, 322–329
 - replacing, 329
 - self-documenting, 323
 - trademarking, 328
 - National Cybersecurity Strategy, 190
 - National Institute of Standards and Technology (NIST), 187, 194, 260
 - National Telecommunications and Information Administration (NTIA), 194
 - National Vulnerability Database, 187
 - NBT, *see* Next Big Thing
 - negativity bias, 360
 - Netflix, 250, 297, 307, 325
 - Netscape, 8
 - network reciprocal licenses, 48
 - networks, distribution over, 61
 - New York Public Library, 231
 - nexB, 193
 - Next Big Thing, FOSS advantages, 88, 249, 252
 - NIH, *see* Not Invented Here syndrome
 - NIST (National Institute of Standards and Technology), 187, 194, 260
 - non-code, outbound FOSS, 231
 - non-software
 - BOM formats for, 197
 - contributions, 266
 - identifying inbound FOSS, 126
 - releases, 303, 308
 - Nonprofit Explorer, 343
 - nontriviality, copyright and, 27
 - Not Invented Here syndrome
 - avoiding, 140–142
 - FOSS advantages, 88
 - perception of FOSS as insecure and, 143
 - NOTICE files, 206, 353
 - notices, 206, 353
 - NTIA (National Telecommunications and Information Administration), 194
 - NumFocus, 343
 - NumPy, 125
 - Nutanix, 205

O

- Objectives and Key Results (OKRs), 302
- onboarding
 - contribution policies and, 274, 277
 - contributors, 287–288, 292
 - outbound FOSS benefits and, 248
 - policies and procedures training, 156
- Open Bioinformatics Foundation, 224
- Open Container Initiative, 88
- open core
 - AGPL and, 49
 - using, 381–384
 - multi-licensing and, 348
 - OSPOs and, 110–111
 - release but retain control approach, 309
 - relicensing and, 72, 383
 - term confusion and, 372
 - using outbound FOSS as loss leader, 251
- Open Data Commons licenses, 55, 303
- Open Invention Network, 23
- Open Knowledge Foundation, 55
- Open Source Collective, 224
- Open Source Definition (OSD)
 - criteria, 9, 50
 - development of, 9
 - importance of standards and, 15
 - OSI certification process, 50–52
- Open Source Hardware Association, 55, 303
- Open Source Initiative (OSI)
 - certification process, 50–52
 - creation of, 10
 - importance of standards and, 15
 - membership in, 224
 - numbers of licenses, 209
- Open Source Law, Policy and Practice*, 424
- Open Source Program Office
 - in academia and public sector, 103
 - community and, 104, 107–113, 116, 361
 - compliance and, 105, 107, 118
 - contributions and, 109, 265–266, 286, 289–294
 - contributions without, 294
 - cross-functionality and, 102–103
 - defined, 101
 - determining need for, 102
 - edge case policy questions, 239
 - functions, 101, 103–107
 - hiring maintainers and, 223
 - isolation of, 112
 - multi-licensing and, 57
 - org chart placement, 117–119
 - outsourcing, 119
 - releases and, 108–109, 111, 356
 - reporting chain, 118
 - roles within, 113–117
 - software composition analysis and, 182
 - sustainability and, 226
 - types of, 107–113
 - vendoring and, 171
- Open Source Review Board (OSRB), 102
- Open Source Review Toolkit (ORT), 193, 208, 212
- open source software, *see also* free and open source software (FOSS)
 - defined, 8
 - development of, 9–11
 - vs. free software, 11
 - standards, importance of, 15
- Open Worldwide Application Security Project (OWASP) Foundation, 197
- OpenCollective, 267
- OpenSearch, 45, 362, 384
- OpenSSF Best Practices Working Group, 260
- OpenSSL, 72, 178
- OpenTofu, 45, 362, 384
- operating systems, identifying inbound FOSS, 126
- operations, BOM formats for, 197
- optimization, premature, 212
- Oracle, 28, 348
- originality, copyright and, 24, 27
- ORT (Open Source Review Toolkit), 193, 208, 212
- OSPO, *see* Open Source Program Office
- OSPO Alliance, 102
- OSPO++ community, 103
- OSRB (Open Source Review Board), 102
- Osterwalder, Alexander, 425
- Our Common Future* (Brundtland Report), 213–215
- outbound FOSS, *see also* contributions; releases
 - agile development and, 253–255
 - benefits, 232, 247
 - corporate strategy and, 131, 243–261
 - costs, 245
 - defined, 229
 - goals, clarifying, 243–246
 - impact on organization, 229–231
 - initiative value, showing, 244
 - non-code, 231
 - policies, 256, 261
 - postponing policies and procedures, 241
 - procedures, 240, 256, 261
 - risks, 75–86, 255–261
 - stakeholders, 229
 - types of, 231
 - using as loss leader, 251
- outreach, OSPOs and, 110
- Outreachy, 95, 358
- OW2, 332, 343
- OWASP (Open Worldwide Application Security Project Foundation), 197
- ownership
 - copyright, 29–30, 236
 - policies and procedures, 160
 - total cost of ownership, 89
- O'Brien, Duane, 268

P

- paid internships, 358
- Paris, 103

- patents
 - algorithms, 26
 - applications, 21
 - basics of, 21–23
 - community and, 23, 276
 - contributions and, 275
 - jurisdiction and, 21
 - licenses and, 21–23
 - OSPO type and, 108
 - patent clauses in licenses, 22, 49, 57, 275, 346
 - patent pools, 23, 276
 - patent reviews, 275, 330
 - patent trolls, 23
 - release preparation and, 330
 - uniqueness and, 21
- permissive licenses
 - defined, 49
 - license compatibility and, 62
 - license selection and, 345, 347
- personally identifying information (PII), security and, 337
- Pigneur, Yves, 425
- Plex, 231
- policies
 - antitrust and anticompetition policies, 300
 - automating approvals, 165
 - automating enforcement, 156, 159, 211
 - build-versus-buy question, 140
 - as business assets, 146
 - checklists, 159
 - collaboration and, 154
 - commodity, 210, 212
 - contributions, 169, 171, 235–237, 273–274, 277, 287, 292
 - Contributor License Agreements, 37, 236, 274
 - defined, 146
 - deploying, 155
 - Developer Certificates of Origin, 37, 236, 274
 - developer input on, 128
 - disaster planning, 168
 - edge cases, 238
 - enforcing, 147, 156–160, 193, 257
 - escalation path, 158
 - evolving, 160, 239
 - examples, 239
 - explaining reasoning behind, 148, 155
 - forking, 171
 - FOSS-first, 141, 171
 - gathering information, 154
 - help information, 235, 239
 - importance of, 145
 - inbound FOSS, 140, 163–171, 209–212, 233–241
 - information gathering, 148
 - IP, 163–166, 233–240, 256, 274
 - license compliance, 209–212
 - licenses, 52, 209–212
 - lists of, 240
 - maintenance of, 153, 160, 165
 - making policies open source, 237
 - micromanaging, 145–146
 - neglecting, 145–146
 - open-first, 171
 - outbound FOSS, 256, 261
 - postponing, 241
 - premature optimization, avoiding, 212
 - releases, 235, 238, 353
 - removing, 161
 - security, 166–168, 261
 - standardizing format, 148
 - storing and access to, 151–152, 159
 - strategies for, 147–155
 - sustainability, 168
 - trademarks, 328–329
 - training and onboarding on, 156
 - vendorship (integration), 169–171
 - verifying, 277, 353
 - version control, 151, 160
 - what to include, 153
 - writing, 147–153
- Porsche, 297
- premature optimization, 212
- Principle of Least Surprise, 365
- privacy
 - Contributor License Agreements, 35
 - outbound FOSS risks, 260
- procedures
 - automating approvals, 165
 - automating enforcement, 156, 159
 - as business assets, 146
 - checklists, 159
 - collaboration and, 154
 - contributions, 241, 287, 292
 - Contributor License Agreements, 241, 274
 - defined, 145
 - deploying, 155
 - Developer Certificates of Origin, 241, 274
 - developer input on, 128
 - disaster planning, 168
 - enforcing, 147, 156–160
 - escalation path, 158
 - evolving, 160
 - gathering information, 154
 - importance of, 145
 - maintenance of, 153, 160, 165
 - micromanaging, 145–146
 - neglecting, 145–146
 - outbound FOSS, 240, 256, 261
 - postponing, 241
 - releases, 241, 353
 - removing, 161
 - standardizing format, 148
 - storing and access to, 151–152, 159
 - strategies for, 147–155
 - training and onboarding on, 156
 - verifying, 353
 - version control, 151, 160
 - what to include, 153
 - writing, 147–153
- Producing Open Source Software*, 317, 335, 424
- productivity, outbound FOSS and, 232
- products
 - FOSS brand reputation benefits and, 91
 - market research, 230, 248
 - outbound FOSS benefits, 230, 252–255
 - product management team as FOSS stakeholder, 128, 230

- product-focused OSPOs, 111, 118
 - release benefits, 300
 - profanity, 331
 - professional development, 94, 232, 248
 - Program Management for Open Source Projects*, 115, 335, 355, 361, 423
 - project health, *see also* sustainability
 - metrics, 74, 221
 - risks and, 74
 - as sustainability indicator, 220
 - project/program managers and OSPO, 115
 - projects, *see also* releases
 - archiving, 366
 - closing, 366
 - donating to foundations, 311, 320, 342, 346
 - identifying critical projects for contributions, 285, 290
 - identifying project needs, 285, 291
 - leaving, 364–367
 - naming, 81, 322–329
 - preparing for release, 329–336
 - project availability, 306
 - sponsorships, 224, 266
 - website and blog, 356
 - promotion
 - contributions, 288, 292
 - OSPO and, 105
 - releases, 336, 354–359
 - proprietary licenses
 - acquisition of proprietary software, 176
 - complexity of, 43
 - uses, 55
 - vendors and, 177
 - public domain, 33, 54
 - purl, 193
 - Python Software Foundation, 224, 276
- R**
-
- README files, 334, 349, 366
 - reciprocal licenses, *see also* AGPL
 - cryptographic keys, 46
 - defined, 44
 - dependencies and, 82
 - derivatives and, 46–47, 59
 - development of, 7
 - fear of, 46, 166, 203
 - free software vs. open source software, 12
 - license compatibility and, 62
 - license compliance and providing source code, 208, 211
 - network, 48
 - selecting, 345–348
 - sense of community and, 90
 - strong, 46–47
 - weak, 48
 - recruitment
 - from communities, 92, 109, 247
 - contribution metrics, 281
 - diversity and, 96
 - FOSS advantages, 92–96, 232
 - OSPO and, 109
 - outbound FOSS and, 230, 247
 - Red Hat, 377, 380
 - Red Hat Enterprise Linux, 377
 - Red Hat Linux, 377
 - redict, 384
 - Redis, 362, 384
 - regulations, *see also* cybersecurity
 - export, 259, 305
 - OSPO and, 108
 - outbound FOSS risks, 258
 - releases, *see also* forges
 - benefits of, 249–255, 298–301
 - checks, prerelease, 353
 - community management and, 85, 360
 - community and release approaches, 310–313, 320
 - complying with component licenses, 353
 - contribution guidelines, 334
 - deciding on whether to release, 303–305, 312
 - documentation, 333–336
 - donating projects to foundations, 311, 320, 342, 346
 - enabling project evolution, 363
 - examples of, 297, 313–316
 - goals, 301, 306
 - governance and, 310–313, 320, 332
 - language cleanup, 331
 - leaving projects, 364–367
 - license compliance and, 82, 353
 - licenses, 82, 344–353
 - locations, 340–342
 - maintenance costs, 83
 - messaging on, 336, 354–359
 - metrics, 306, 340, 359
 - naming projects, 322–329
 - non-software, 303, 308
 - with open governance, 310–313, 320
 - OSPO and, 108–109, 111, 356
 - overview of steps, 317
 - permissive licenses and, 49
 - policies, 235, 238, 353
 - preparing projects for, 329–336
 - procedures, 241, 353
 - project availability and, 306
 - project management and, 361
 - project types, 305–308
 - project website and blog, 356
 - readiness of, 332
 - release but retain control approach, 309, 320, 356
 - releases without licenses, 67, 82
 - reviews, 78
 - risks, 75–86, 255–261, 309, 315
 - sanitizing and securing, 329–332
 - security, 77–79, 330, 334
 - strategies for, 297–316
 - strategy considerations, 75–77, 98
 - style worksheet, 321, 422
 - styles of, 308–313, 320
 - team, 318–319

- throwing it over the wall, 13, 84, 298, 308, 320
 - tool compatibility and, 308
 - trademarks, 20, 80–82, 324, 329
 - when not to release, 322
 - relicensing
 - Contributor License Agreements and, 383
 - license selection and, 347
 - open core and, 72, 383
 - risks, 72
 - to non-open source, 72
 - remediation, software composition analysis, 189–191
 - repair, right to, 205
 - reputation, *see* brand reputation
 - research, OSPO and, 106
 - resilience, diversity and, 95
 - Responsible Communication Style Guide, 326
 - retention
 - contribution metrics, 281
 - diversity and, 95
 - FOSS advantages, 93, 95, 232
 - outbound FOSS and, 247
 - REUSE, 346, 351
 - revenue models, 371, 375
 - right to repair, 205
 - RISC-V, 252
 - risk, *see also* liability
 - community and, 85
 - contributions and, 30–33, 255–261
 - Contributor License Agreements, 35
 - high risk profile, 165
 - inbound FOSS, 65–75, 143
 - integrating FOSS into business models, 379–381
 - IP, 75–86, 164, 256
 - license compliance and, 67, 82, 211
 - low risk profile, 165
 - maintenance, 83
 - OSPO type and, 108
 - outbound FOSS, 75–86, 255–261
 - releases, 75–86, 255–261, 309, 315
 - relicensing, 72
 - risk profile and contributions, 273
 - risk profile and license compliance policies, 211
 - risk profile and software composition analysis, 180, 183
 - software supply chain, 65–75, 78, 96
 - sustainability, 74, 129
 - trademarks, 80–82
 - Roads and Bridges*, 217, 267
 - Rothman, Johanna, 365
 - Rust, 57
- ## S
-
- SaaS (software-as-a-service)
 - AGPL and, 48, 61, 300, 346
 - distribution and, 61
 - SSPL and, 56
 - sabotage, 168, 179, 192
 - SAFECodes, 260
 - sales
 - FOSS brand reputation benefits and, 91
 - OSPOs and, 111
 - sanctions, 259
 - SBOMs
 - containers, 209
 - contents, 195
 - formats, 195–198
 - notice generation, 208
 - potential of, 187, 198
 - releases and, 346
 - SPDX license identifiers and, 351
 - using, 194–199
 - SCA, *see* software composition analysis
 - ScanCode, 193
 - scanning, software composition analysis, 182–186
 - SCO Group, 339
 - SDGs (Sustainable Development Goals), 214
 - security
 - best practices, 78, 260
 - containers and, 69
 - contributions and, 97, 278
 - Contributor License Agreements, 35
 - disaster planning, 168
 - diversity and, 97
 - EO 14028, 179, 192, 194, 258
 - FOSS advantages, 96–98, 143
 - FOSS risks, 143, 260
 - information security, 260
 - National Cybersecurity Strategy, 190
 - through obscurity, 69
 - OSPO and, 108
 - perceptions of FOSS as insecure, 143
 - personally identifying information and, 337
 - policies, 166–168, 261
 - preparing projects for release, 330
 - releases, 77–79, 330, 334
 - review of contributions and, 97
 - security team as FOSS stakeholder, 127
 - security team and outbound FOSS, 230
 - software supply chain awareness and, 178
 - software supply chain risk, 66, 68–70, 78, 96, 129
 - sustainability and, 129, 218
 - vendoring, 170
 - vulnerability databases, 187
 - SECURITY.md files, 334
 - Sentry, 247
 - Server Side Public License (SSPL-1.0), 56, 72
 - shadow IT, 177
 - shifting left, 181
 - Simian Army, 250, 325
 - single-vendor commercial FOSS, 111, 309, 348
 - skills
 - constraints and release approaches, 320
 - constraints on contributions, 284
 - SLSA (Supply-chain Level for Software Artifacts), 330
 - SMART goals, 302
 - Snipe-IT, 379–380
 - snippets, software composition analysis, 183
 - software composition analysis
 - analyzing step, 186–189

- automating, 181, 188, 191
- containers, 184
- defined, 179
- documenting, 188
- focus on critical links, 188, 219
- frequency of, 191
- manual research, 180
- notice generation, 208
- process, 182–192
- remediation, 189–191
- repeat step, 191
- risk profile and, 180, 183
- scanning, 182–186
- services, 186
- sustainability and, 219–222
- tools, 183, 192–194, 209
- versions and, 187
- Software Freedom Conservancy, 203, 205, 223–224, 267, 346
- Software Heritage, 76, 185
- Software in the Public Interest, 267, 343
- software supply chain, *see also* software composition analysis
 - awareness of, 129, 173, 178, 219
 - defined, 129, 175
 - as ecosystem, 217
 - ecosystems and sustainability, 215–217
 - FOSS acquisition types, 176
 - identifying inbound FOSS, 125
 - liability and, 70
 - license compliance inspection, 201
 - license compliance risks, 67, 82
 - license compliance verification, 208
 - links in as having own supply chains, 176
 - OSPO and, 105, 108–109, 114
 - risks, 65–75, 96
 - security, 66, 68–70, 78, 96, 129
 - strategies for, 175
 - sustainability and, 213–226, 246, 263
 - software-as-a-service, *see* SaaS (software-as-a-service)
 - source code
 - copyright and license statements, 351
 - as copyrightable, 26
 - fauxpen source licenses, 56
 - license compliance and providing, 208, 211
 - problematic code and searching code snippets, 183
 - SPDX, 195, 198, 351
 - SPDX License List, 349
 - sponsorships
 - fiscal, 267
 - projects, 224, 266
 - SSPL-1.0 License, 56, 72
 - Stallman, Richard M., 6
 - standards
 - in example of a release, 314
 - importance of, 14–16
 - license standardization of FOSS, 43, 52
 - outbound FOSS benefits, 252
 - stop lists, 166
 - supply chain managers, 114
 - supply chains, *see also* software supply chain
 - defined, 173
 - supply webs, 174, 180
 - supply webs, 174, 180
 - Supply-chain Level for Software Artifacts (SLSA), 330
 - Supreme Court, 28
 - sustainability
 - Brundtland Report, 213–215
 - as business matter, 213–217
 - complexity and, 215
 - contribution metrics, 281
 - contributions and, 169, 222, 246, 263
 - defined, 214
 - ecosystems and, 215–217
 - financial support, 223
 - forking and, 362
 - foundation memberships and support, 223
 - integrating FOSS with business model and, 381
 - interdependency and, 215
 - maintainers and, 217, 222
 - maintenance and, 74, 169
 - policies, 168
 - prioritizing, 225
 - project health as indicator of, 220
 - project sponsorships, 224, 266
 - risks, 74, 129
 - security and, 218
 - software supply chain, 213–226, 246, 263
 - state of open source, 217
 - strategies for, 219–226
 - systems thinking and, 215
 - Sustainable Development Goals (SDGs), 214
 - systems thinking, 215

T

 - talent, contributions of, 265
 - target markets and channels in business models, 371, 375, 382
 - Tcl/Tk License, 210
 - technical debt, 190
 - technical marketers, 115, 356
 - technical writers
 - documentation and, 335
 - as FOSS stakeholders, 128
 - OSPO and, 116
 - policies and procedures, 152
 - termination clauses, 204
 - Tern, 194, 209
 - Terraform, 45, 362, 384
 - Thengvall, Mary, 425
 - think tank-type OSPOs, 112, 118
 - throwing it over the wall releases
 - community and, 13
 - documentation and, 335
 - maintenance and, 84, 298, 308
 - release preparation, 320

- ticketing
 - Getting Started Guides, 288
 - policy questions, 239
 - time
 - constraints and release approaches, 320
 - constraints on contributions, 284
 - contributions of, 264
 - time to market, FOSS advantages, 88, 141
 - TINLA, 18
 - TODO Group, 102, 240
 - tooling developers, 115
 - tools
 - about, *xxi*
 - agile development and, 254
 - inbound FOSS and, 125
 - notice generation, 208
 - policies and procedures and, 150
 - releases and tool compatibility, 308
 - software composition analysis, 183, 192–194, 209
 - total cost of ownership, 89
 - trademarks
 - contributions and, 276
 - defined, 19
 - distinctiveness and, 20
 - integrating FOSS with business model and, 380
 - jurisdiction and, 20
 - mergers and acquisitions, 81, 324
 - name, 328
 - policies, 328–329
 - registering, 20, 80
 - releases, 20, 80–82, 324, 329
 - transformation and fair use, 42
 - transparency
 - foundations and, 343
 - governance and, 333
 - trolls
 - copyright trolls, 203
 - patent trolls, 23
- ## U
- uniqueness, patents and, 21
 - United Nations, 213–215
 - United States Copyright Office, 24
 - United States Executive Order on Improving the Nation's Cybersecurity, *see* EO 14028
 - Unity Reference-Only License, 55
 - Unity Technologies, 55
 - UNIX, copyright lawsuit, 339
 - upstream
 - contributions, 171, 270–273
 - development, 269–273
 - vendoring and, 170
 - utilities, sustainability and, 220
- ## V
- Valkey, 362, 384
 - value proposition in business models, 371, 374
 - vendoring, 169–171
 - vendors vs. maintainers, 177
 - version control
 - Developer Certificates of Origin signoffs, 36, 338
 - forges and, 117, 341–342
 - identity management and, 280
 - policies and procedures, 151, 160
 - in software composition analysis, 220
 - versions
 - software composition analysis and, 187
 - vendoring and, 170
 - Vets Who Code, 95
 - VEX documents, 185
 - viral licenses, *see* reciprocal licenses
 - Vizio, 205
- ## W
- wages, 232
 - Wango Worldwide NGO Directory, 342
 - Wardley Maps, 134
 - Wardley, Simon, 134
 - warranty clauses, 41, 70, 79, 178
 - Why Startups Fail*, 379
 - WIPO, *see* World Intellectual Property Organization
 - Women in Linux, 95
 - WordPress, 66
 - work, defined, 24
 - work for hire and copyright, 29
 - World Intellectual Property Organization, 19, 24
 - writers, *see* technical writers
- ## Y
- Yocto, 93
- ## Z
- Zalando, 240

Thank you!

We hope you enjoyed this book and that you're already thinking about what you want to learn next. To help make that decision easier, we're offering you this gift.

Head on over to <https://pragprog.com> right now, and use the coupon code BUYANOTHER2025 to save 30% on your next ebook. Offer is void where prohibited or restricted. This offer does not apply to any edition of *The Pragmatic Programmer* ebook.

And if you'd like to share your own expertise with the world, why not propose a writing idea to us? After all, many of our best authors started off as our readers, just like you. With up to a 50% royalty, world-class editorial services, and a name you trust, there's nothing to lose. Visit <https://pragprog.com/become-an-author/> today to learn more and to get started.

Thank you for your continued support. We hope to hear from you again soon!

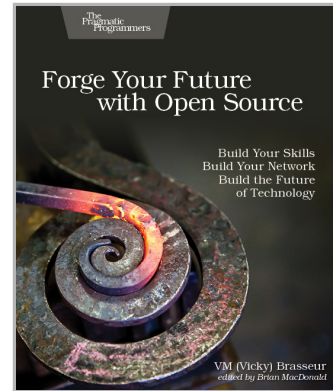
The Pragmatic Bookshelf



Forge Your Future with Open Source

Free and open source is the foundation of software development, and it's built by people just like you. Discover the fundamental tenets that drive the movement. Take control of your career by selecting the right project to meet your professional goals. Master the language and avoid the pitfalls that typically ensnare new contributors. Join a community of like-minded people and change the world. Programmers, writers, designers, and everyone interested in software will make their mark through free and open source software contributions.

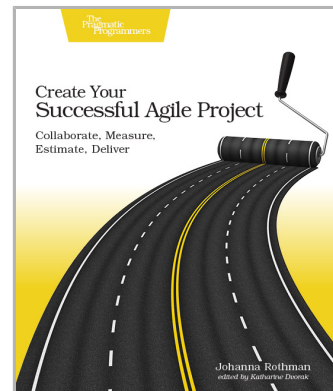
VM (Vicky) Brasseur
(222 pages) ISBN: 9781680503012. \$33.95
<https://pragprog.com/book/vbopens>



Create Your Successful Agile Project

You think agile techniques might be for you, but your projects and organization are unique. An “out-of-the-box” agile approach won’t work. Instead, unite agile and lean principles for your project. See how to design a custom approach, reap the benefits of collaboration, and deliver value. For project managers who want to use agile techniques, managers who want to start, and technical leaders who want to know more and succeed, this book is your first step toward agile project success.

Johanna Rothman
(290 pages) ISBN: 9781680502602. \$45.95
<https://pragprog.com/book/jragm>



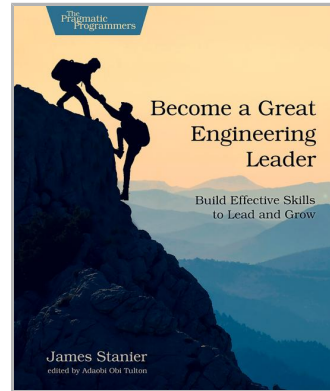
Become a Great Engineering Leader

As you step into senior engineering leadership roles, you need to make an impact, and you need to make it fast. This book will uncover the secrets of what it means to be a successful director of engineering, VP of engineering, or CTO. With a hands-on, practical approach, it will help you understand and develop the skills that you need, ranging from how to manage other managers, to how to define and execute strategy, how to manage yourself and your limited time, and how to navigate your own career journey to your desired destination.

James Stanier

(400 pages) ISBN: 9798888650660. \$64.95

<https://pragprog.com/book/jsenglb>



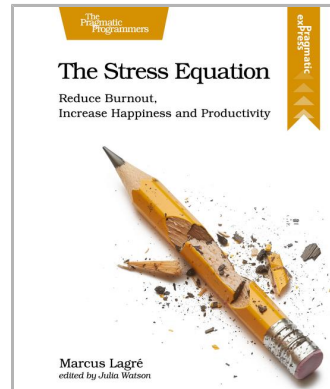
The Stress Equation

Workplace stress is not the weakness of individuals; it's caused by systemic problems. Armed with the insights in this book, you can identify, analyze, and systematically reduce the factors that lead to poor health, low productivity, and personal burnout. This book gives you a framework for understanding stress, and a vocabulary to make it easier to discuss it among colleagues. Stress can be fixed; find out how.

Marcus Lagré

(126 pages) ISBN: 9798888651018. \$35.95

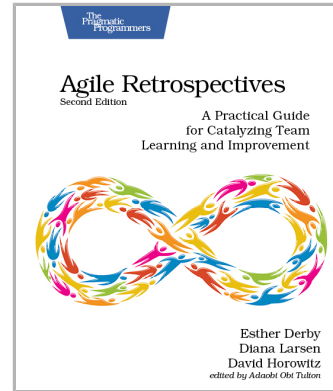
<https://pragprog.com/book/stresseq>



Agile Retrospectives, Second Edition

In an uncertain and complex world, learning is more important than ever before. In fact, it can be a competitive advantage. Teams and organizations that learn rapidly deliver greater customer value faster and more reliably. Furthermore, those teams are more engaged, more productive, and more satisfied. The most effective way to enable teams to learn is by holding regular retrospectives. Unfortunately, many teams only get shallow results from their retrospectives. This book is filled with practical advice, techniques, and real-life examples that will take retrospectives to the next level—whether your team is co-located, hybrid, or remote. This book will help team leads, scrum masters, and coaches engage their teams to learn, improve, and deliver greater results.

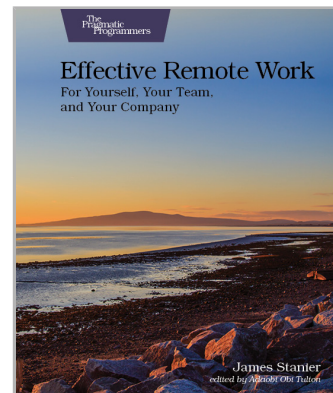
Esther Derby, Diana Larsen, David Horowitz
(298 pages) ISBN: 9798888650370. \$53.95
<https://pragprog.com/book/dlret2>



Effective Remote Work

The office isn't as essential as it used to be. Flexible working hours and distributed teams are replacing decades of on-site, open-plan office culture. Wherever you work from nowadays, your colleagues are likely to be somewhere else. No more whiteboards. No more water coolers. And certainly no Ping-Pong. So how can you organize yourself, ship software, communicate, and be impactful as part of a globally distributed workforce? We'll show you how. It's time to adopt a brand-new mindset. Remote working is here to stay. Come and join us.

James Stanier
(348 pages) ISBN: 9781680509229. \$47.95
<https://pragprog.com/book/jsrw>



The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by professional developers for professional developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

This Book's Home Page

<https://pragprog.com/book/vbfoss>

Source code from this book, errata, and other resources. Come give us feedback, too!

Keep Up-to-Date

<https://pragprog.com>

Join our announcement mailing list (low volume) or follow us on Twitter @pragprog for new titles, sales, coupons, hot tips, and more.

New and Noteworthy

<https://pragprog.com/news>

Check out the latest Pragmatic developments, new titles, and other offerings.

Buy the Book

If you liked this ebook, perhaps you'd like to have a paper copy of the book. Paperbacks are available from your local independent bookstore and wherever fine books are sold.

Contact Us

Online Orders: <https://pragprog.com/catalog>

Customer Service: support@pragprog.com

International Rights: translations@pragprog.com

Academic Use: academic@pragprog.com

Write for Us: <http://write-for-us.pragprog.com>