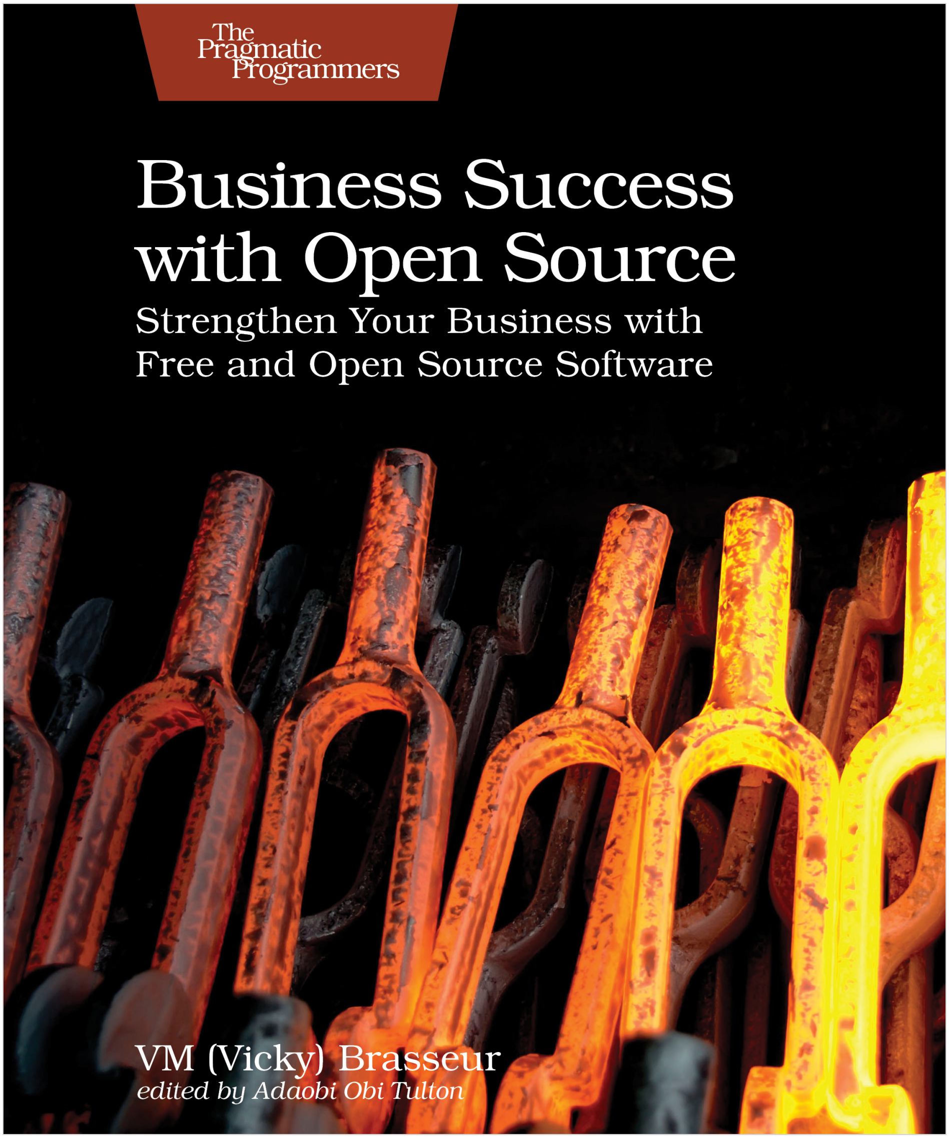
The Pragmatic Programmers

Business Success with Open Source

Strengthen Your Business with Free and Open Source Software

VM (Vicky) Brasseur

edited by Adaobi Obi Tulton

CHAPTER 4

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 ?, 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 ?, 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 ?.

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 ?, 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

- *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.
- *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 ?
- Chapter 12, Maintain FOSS License Compliance, on page ?

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 ?
- Chapter 12, Maintain FOSS License Compliance, on page ?
- The Container Complication, on page ?

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 ?
- Chapter 12, Maintain FOSS License Compliance, on page ?