THE LINUX FOUNDATION

Fact gathering: the first and most important task in software negotiations

Karen Copenhaver and Steve Winslow

www.linuxfoundation.org

Table of Contents

Introduction 3

Software is not static 4

A software provider will not be the author of all of the software that is being delivered 5

Software will be developed with a set of tools, which can be important to the delivery of software and solutions 7

Many of the most valuable third-party components and tools are made available under open source licenses 9

Software licenses can be categorized in unlimited ways 10

Some of the most essential and widely used software is provided under the GPL and other copyleft licenses 11

Conclusion 12

Introduction by Karen Copenhaver:

When I began practicing law 30 years ago, I had a fabulous manager. He always reminded all of us that practicing law is ninety percent fact gathering and ten percent legal analysis. You cannot do the legal analysis until you fully understand the facts. If our drafting was vague, it was almost always because we were writing around facts that we didn't fully understand. We were leaving space because we could not be precise.

What is true of individual lawyers is also true of groups of lawyers and procurement professionals attempting to negotiate an agreement. If both parties shared a common understanding of the facts, drafting the agreement would be much more efficient. By facts, I mean details about the context of the agreement that are fixed and verifiable. I am not suggesting that the parties could or need to share opinions or a shared vision of how the relationship might unfold. I am thinking about negotiators who do not know what the human beings employed by both parties who will perform the work all know to be true.

As an easy example, when the people negotiating a software development agreement do not know that the developers for both parties assume that the software will include many pre-existing components, the process will be horribly inefficient. This is more than just a waste of time. When the developers are confronted with ridiculous assumptions about writing software from scratch, the credibility of the procurement process is undermined, and, in the future, they will find ways to avoid or delay involving procurement and counsel. Where the parties write around what they do not understand, the agreement that results will be full of vague or inaccurate language.

The purpose of this article is to lay out some basic facts about how software is developed and works today in an attempt to help procurement professionals and their legal counsel avoid making factual assumptions that will undermine their credibility and delay negotiations.

Software is not static

The software that will be developed will evolve as it is developed.

In the early days of the software industry, attorneys assumed that they could comprehensively capture a business relationship in the written contract. The agreement would include detailed specifications for the product to be developed, and there would be a project timeline with milestones tied to achieving those specifications. When the agreement was executed, usually after many months of negotiations, the parties knew exactly what they were contractually obligated to deliver and when. Even then, this was a myth.

If we required a detailed, final specification for development before the parties could begin work today, we would only be assured that the results would be too late to market to be valuable. Agreements today have to be focused on establishing the process for working together to develop something which neither party can fully define or envision. In other words, the only thing we know is that what we develop together will change as the work is performed, the operating environment is updated, and the market changes.

Thus, requiring that a fixed list of the specific software components that will be used in the development be included in the agreement may not make sense to the people who will actually perform the work. They may know that the list will change often, and they do not want to amend the agreement every time they consider, include, or replace a component. A process acceptable to both parties that allows for the rapid evolution of the work to be performed will be welcomed.

Software will change continuously over the course of its normal life.

Software is never "finished" until it is uninstalled. Constant updating is required to accommodate changes in the operating environment and to apply patches that become available to eliminate potential security vulnerabilities. If the software is not updated, that should be a sign that necessary software maintenance is not occurring. And changes in the hardware or software operating environment provide opportunities to improve software functionality. The agreement should not be written based on the assumption that all development will come to a conclusion at any point prior to the end of the life of the software.

A software provider will not be the author of and will not "own" the copyright in all of the software that is being delivered

A company that provides software will almost never be the sole author, nor the sole copyright owner, of the entirety of what they provide to the recipient. Software will include components owned and developed by third parties and will rely on dependencies that may not be part of the distributed package of software.

Developers do not sit down to develop software with a clean sheet of paper – just as lawyers do not sit down to draft agreements from scratch. They make use of templates and libraries both for efficiency and for efficacy. Software libraries that have been in use for many years and have been deployed for many purposes benefit from the fixes and improvements provided by others. As a highly regarded technical expert said in a negotiation, "Believe me. You do not want anyone writing a new math library from scratch so that you can own it."

Because software does not operate in a vacuum, components, and interfaces written by third parties are necessary for the software to function. For example, applications installed on a laptop use interfaces in the operating system. The functionality provided over a network sits on top of a stack of software that is so ubiquitous its value is rarely acknowledged. Without using the libraries and/or interfaces that provide access to this infrastructure, the software cannot be developed, tested, or deployed.

In addition to what is developed and delivered as part of the agreement, all software operates within one or more ecosystems of third-party dependencies that are necessary for its optimal use and performance.

This was the case even when software was something that was typically purchased in physical form and installed from a floppy disk or CD-ROM: the software's packaging would list its minimum system requirements, which could include hardware, software, services (such as sufficient Internet bandwidth) and more. The purchaser reasonably needs to know what those dependencies are. Still, the purchaser might not reasonably expect that the provider is going to make contractual commitments regarding the entire stack of those dependencies.

In modern software ecosystems, the situation is exponentially more complicated. Leveraged dependencies might be needed when the software is

built, or when it is installed, or when it runs. The software provider might not deliver these dependencies to the purchaser at all but might instead provide only a manifest file listing the dependencies or a recipe for how to install and configure the dependency environment. As part of the installation process, the purchaser would use these manifests and recipes to obtain those dependencies directly from the upstream third parties that make them publicly available.

The use of these dependencies will directly influence the price at which the software provider offers their software for sale. If the software provider was contractually required to be the original author or copyright holder of all relevant code that is utilized by the software, then the price would be astronomically higher because every software product would require starting from scratch and disregarding the ecosystems of established, well-tested, pre-existing code.

Software will be developed with a set of tools, which can be important to the delivery of software and solutions

Just as lawyers rely on a word processing program to write an agreement, software developers use software tools to make development more efficient. These tools are often the most complex software involved in the development project and the amount of code in this development environment will almost always far exceed the amount of code in the developed deliverable. And this software will change and evolve just as the software that is being developed will change and evolve. Knowing the specific facts related to the collection of tools used to develop this software is essential to avoid unworkable approaches.

Sometimes the development environment will be a third-party product that can be acquired directly from the third party. If a version of the third-party product that is being used is specified, the customer will be able to replicate and maintain that development environment should it ever be needed.

Other times, the reason to hire a specific company to do the work is that they have a well-established, unique development environment, and, just as important, a set of highly skilled developers trained to use it.

These tools operate within their own very complex operating environment. They are not like hammers and screwdrivers that can be put into a box and used separately. The phrase that is commonly used is “development environment” because the tools are integrated into a complex ecosystem and are not useful or necessarily trustworthy outside of that ecosystem.

To “deliver” the entire development environment is often impractical for a number of reasons. The company asking for it to be delivered may not have sufficient equipment or technical employees even to install the software, much less maintain it. And no one would deploy software developed by individuals at the bottom of a learning curve. In one negotiation that was hung up on a demand to deliver all of the tools used in development, a technical person employed by the company making the demand, when asked to weigh in on the request replied, “We wouldn’t know what to do with it if we had it.”

To deliver any code at a single point in time, without a plan for someone to maintain the code going forward, is not useful. The following day that software may be

dangerously out of date. Similarly, establishing a source code escrow arrangement may be a significant amount of effort for very little practical risk mitigation. Access to some version of some amount of source code will not be of much practical use to a purchaser who does not have the specifically-configured development environment in which it was built or experience with how to build and deploy it. Deciding to put an entire development environment, the hardware, software toolchain, and source code into escrow would be prohibitively expensive.

Delivering the code in a development environment is an enormous amount of work that must be performed by highly skilled individuals - often the same individuals who are required to perform the work you have engaged the vendor to perform. Requiring delivery of the development environment as a contract solution where the technical employees of both companies know that the delivered code will never be used, is experienced by developers as a frustrating waste of valuable resources that will delay the work everyone wants the vendor to perform.

Many of the most valuable third-party components and tools are made available under open source licenses

It is almost never possible to fulfill a contractual requirement not to use open source software in development. There are extremely valuable compilers and other development tools that are used in essentially all software development environments that are made available under open source licenses. Depending on the analyst firm, most estimates suggest 70-90% of all the code in a system will be built from open source software. And even proprietary, purchased solutions that your technical team currently uses are very likely built in large part with open source components. Unless your own technical people agree that there should be no open source code of any kind used in its development, do not ask for a representation or warranty that there will be no open source from a contractor or supplier.

If software made available under an open source license will be used, the relevant questions you should ask relating to the selection of the code, maintenance of the code, and compliance with the applicable license terms in the specific use case. And all of these are questions that should be asked about both open source and non-open source software.

If all of your competitors are using these valuable open source assets, and you do not, it will be difficult to be competitive on cost, quality, maintenance, and security. One of the most important reasons to use open source is to benefit from the advantages of shared support across an ecosystem.

Software licenses can be categorized in unlimited ways

There are many software licenses. Some are licenses that the Open Source Initiative (**https://opensource.org/**) has approved as consistent with the Open Source Definition (**https://opensource.org/osd-annotated**). Some are licenses that are similar to those licenses but have never been approved. Some are sufficiently different from those licenses that they would not be considered by people familiar with this terminology to be "open." Others are clearly what would be generally referred to as "commercial" or "proprietary" licenses. In other words, there is a broad spectrum. And there are more licenses to put on that spectrum every day. The SPDX License List (**https://spdx.org/licenses/**) has been curated by lawyers working in the open source ecosystem and identifies many of the licenses that frequently come up in reviews and negotiations.

The question is: does any practical difference arise in any specific contractual context based on exactly where a license falls on that spectrum? In every context that I can think of, contractual concerns regarding the license applicable to third party software components (selection of the code, maintenance of the code, and compliance with the applicable license terms) will be the same regardless of where that license falls on any spectrum of license types. Spending time and energy trying to define a separate category of Open Source Software is not helpful in reaching an agreement. This will become more important if the licenses for some essential third-party components no longer seek OSI approval. Aside from how the open source ecosystem may categorize licenses, all software licensed from third parties should be evaluated under the same criteria for your project.

Because of the wide variety of licenses with similar effect but minor variations in wording, it may be unintentionally detrimental to require that only OSI-approved licenses may be used for all dependencies or components.

Some of the most essential and widely used software is provided under the GPL and other copyleft licenses

In modern computing, a great amount of the most valuable and useful software components and tools are made available under a version of the Free Software Foundation's General Public License (the "GPL") or another license that is commonly referred to as a copyleft, reciprocal or sharing license.

GPL-licensed software such as the GCC Compiler and the Linux operating system is used by the vast majority of companies and industries around the world. Contrary to urban legend, it is not impossible for commercial companies to comply with copyleft obligations. It is not impossible to use both copyleft software and independent software that is not subject to the copyleft obligations. Companies do this in careful compliance with the license requirements every day all around the world.

The distribution of the software usually triggers copyleft obligations to provide source code. Many businesses are built on top of the GPL-licensed Linux operating system and other copyleft software that is used in the business to provide services but not distributed.

The perception of the GPL and its variants as being unworkable open source licenses is also inaccurate. Keep in mind that the GPL, like all free and open source licenses, does not restrict your usage. As a recipient of GPL software, you have far more expansive license rights to use the software than you have under a proprietary software license agreement. Compliance with the GPL upon a redistribution of the code may be a factor to consider. Still, it is unlikely that you would have the right to redistribute any proprietary software at all. If your technical people are certain that you will not be redistributing the GPL code, then negotiating for a "no GPL allowed" provision in an agreement where you are acquiring software is essentially negotiating to receive fewer rights than you otherwise might have.

Unless your technical people agree that there should be no GPL or copyleft licensed code of any kind used in its development or provided in the work product, do not ask for a representation or warranty that there will be no copyleft software. Once again, the relevant questions related to the selection of the code, maintenance of the code, and compliance with the applicable license terms in the relevant use case.

Conclusion

If we only can make one point, it is that lawyers and procurement professionals should not even attempt to dictate how software development will be accomplished. If negotiations hit a rough patch, take the time to confirm that the real issue is risk allocation. Make sure that the dispute is not due to insistence on facts that your technical team does not believe to be true. This is particularly difficult when longstanding corporate policies are out of step with current realities.

A company can have a "no GPL policy." Still, it cannot operate in most industries without dependence upon the Linux operating system, which is GPL-licensed software. Relying on the policy as an all-powerful argument does not change that fact, nor does it add any benefit if the policy does not reflect the reality of your developers' actual technical operations. Taking the time to gather the facts so you can work from the same knowledge-base as those of your own employees who will actually do or oversee the work to be performed will save time and result in a better agreement and relationship.

The Linux Foundation promotes, protects and standardizes Linux by providing unified resources and services needed for open source to successfully compete with closed platforms.

To learn more about The Linux Foundation or our other initiatives please visit us at **www.linuxfoundation.org**