EDITOR DIRK RIEHLE
Friedrich Alexander-University of Erlangen Nürnberg;
dirk.riehle@fau.de

The Innovations of Open Source

Dirk Riehle, Friedrich-Alexander-Universität, Erlangen-Nürnberg

Open source has given us many innovations. This article provides an overview of the most important innovations and illustrates the impact that open source is having on the software industry and beyond.

FROM THE EDITOR

Welcome to this new column on open source! The "Open Source Expanded" column will aim to provide an insightful article every two months. These articles will be written for the software practitioner by authors from both academia and industry. Articles will be grouped by theme rather than appearing in arbitrary order. Our first theme is about open source licenses and license compliance. Even decades after open source was created, it is still a hot topic and unknown territory for many. Later themes will focus on using open source, project communities, business models, interesting cases, and what might come after open source. This issue's article, the first in its series, provides an overview of what is to come by reviewing the most important innovations that open source has provided for the software industry and beyond. If you have comments or would like to suggest future themes and articles, feel free to contact me at dirk@riehle.org. *Computer* will provide a discussion board for articles as well. *— D. Riehle*

The main innovations of open source can be grouped into four categories: legal, process, tool, and business models. Probably the best known innovations are open source licenses, which also define the concept. Software becomes open source if users receive it under an open source license. To be an open source license, it must fulfill 10 requirements set forth by the Open Source Initiative, the protector and arbiter of what constitutes open source.[1] Most notably, the license must allow

- free-of-charge use of the software
- access to and modification of the source code
- the ability to pass on the source code and a binary copy.

Before there was open source software, there was free software. Richard Stallman defined the four freedoms of software that make it "free" as:[2]

Digital Object Identifier 10.1109/MC.2019.2898163
Date of publication: 16 April 2019

> *the freedom to run the 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 […], the freedom to distribute copies of your modified versions to others […].*

Open source software and free software, and the people behind them, have struggled with each other at times. For all practical purposes, however, the difference is irrelevant to users. What matters is the license under which a user receives a particular software.

LEGAL INNOVATION

Licenses can be structured into permissions (the rights granted to a user), obligations (what is required to receive these rights), and prohibitions (what may not be done; for example, claiming that using the software implies an endorsement by its creator). The two legal innovations are

1. the rights grant as introduced earlier
2. a particular obligation called *copyleft*.

The rights grant helped open source spread and succeed. As research has shown, it taps into the human desire to help each other and collaborate on interesting projects.

People sometimes ask why developers do not put their work into the public domain. This misses the point: by putting something into the public domain, an author typically waives his or her rights, and most authors do not want that. Rather, they want to be specific about which rights they grant and which obligations they require.

The most famous license obligation is probably the copyleft clause. Stallman invented this clause, and it became popular through GNU General Public License v2 in 1991. It states that if you pass on copyleft-licensed code, such as part of a product that you sell, you must also pass on your own source code if it modifies the copyleft-licensed code. The specifics of this can get complicated quickly, and they will be discussed in more detail in future columns. Many companies worry that if their source code is mixed with copyleft-licensed code, they will lose their intellectual property and, hence, their competitive advantage in the marketplace.

In the past, companies have used this clause to incorrectly discredit open source software as "a virus" or "cancer" and a "communist" or "hippie undertaking." However, nobody forces anyone to use open source software. In an amazing about-face, some of the most well-known companies that fought open source only 15 years ago are now among its biggest supporters. The "Business Model Innovation" section of this article explains some of this.

PROCESS INNOVATION

The next innovation open source has brought us is engineering process innovation.[3] The open source initiative has this to say about open source software development:[1]

> *Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.*

This is the other definition of open source, which does not focus on licenses and intellectual property but, rather, on collaborative development. There is no single open source software engineering process because each open source community defines its own.

Through his development of the Linux kernel, Linus Torvalds was the first to explore, at scale, a truly collaborative open source process. His approach has no particular name but is often identified with his moniker, BDFL (which stands for "benevolent dictator for life"), implying a hierarchical structure. A core benefit of an open collaboration process was named after Torvalds and is called *Linus' law*, which states, "Given enough eyeballs, all bugs are shallow."[4] The idea is that more broadly used software matures more quickly since problems are found and solved more quickly.

The collaborative peer group, as explored by the original Apache web server team (httpd) and codified as The Apache Way (of open source software development), is a similar but different approach that may be more popular today.[5] The software industry owes this group of developers as much as it owes Torvalds, if not more.

The Apache Way is a consensus-based, community driven governance approach to collaboration in open source projects. The Apache Software Foundation's website explains it in detail. An important aspect is the distinction between contributors, who submit work for inclusion in an open source project, and committers, who review and integrate the work. Committers are called *maintainers* in a Linux context, and they usually are developers, too, not just reviewers. Using this contributor–committer interplay, nearly all open source projects practice precommit code review to ensure the quality of the software under development.

The principles of open source software development can be summarized as three principles of open collaboration.[6]

- In open collaboration, participation is egalitarian (nobody is a priori excluded).
- Decision making is meritocratic (based on the merits of arguments rather than status in a corporate hierarchy).
- People are self-organizing (they choose projects, processes, and tasks rather than being assigned to them).

Similarly, open source projects practice open communication. This form of communication is public (everyone can see it), written (so you don't have to be there when words are spoken), complete (if it wasn't written down, it wasn't said), and archived (so that people can look up and review discussions later).

Such open collaborative processes, which are not dominated by any single entity, lead to community open source software, which is collectively owned, managed, and developed by a diverse set of stakeholders. These collaboration processes are not limited to software but spill over into adjacent areas. For example, they have brought forward many formal and de facto standards that the software industry relies on.[3] The methods for open source software development have also taken root inside companies, where they are called *inner source*.[7,8]

TOOL INNOVATION

Most of the tools used in open source software development are familiar to closed source programmers as well. However, the needs of open source processes have led to two major tool innovations that have since become an important part of corporate software development as well: software forges and distributed version control.

A software forge is a website that allows the creation of new projects and provides developers with all of the tools needed for software development, such as a home page, an issue tracker, and version control. What makes software forges special is that they facilitate matchmaking between those who are looking to find a useful software component and those who are offering one. They are an enterprise software product category because, even within one company, you want to have one place for all software being developed.

Distributed version control is version control in which you copy the original repository and work with your copy. Thus, you do not need commit rights or ask for permission to start work. Git and Mercurial are the two best-known examples of such software. Some may argue that distributed version control is not an open source innovation because some of its roots are in proprietary software. However, the open source community developed and refined its own solutions, which work well with how open source software is developed, and thereby popularized the concept.

Comparing distributed version control with branching misses the point. Having your own repository allows developers to work using their own style, free of any centralized decisions on how to use branches.

Distributed version control was popularized by being the main version control software underlying a new generation of software forges, most notably Github and Gitlab. As such, companies are adopting both forges and distributed version control at a rapid pace.

BUSINESS MODEL INNOVATION

Open source is changing the software industry by how it makes new business models and breaks old ones. For instance, it lays the legal foundation for open collaboration between individuals and companies, defines more effective collaboration processes with higher productivity than closed-source approaches, and invents the tools to support it. Open source itself may not be a business model, but it is a potent strategy and a tool to use in competitive environments.

For-profit models

There are different approaches for classifying business models enabled by open source, but I like to put them into five categories. Three are for-profit business models, and two are nonprofit models. The for-profit business models are as follows.

1. *Consulting and support business models*: In this conventional model, a company earns money by providing consulting and support services for existing open source software. They do not sell a license, but they service the software anyway. The original open source service company was Cygnus Solutions, which serviced the GNU set of tools. More recent examples are Cloudera and Hortonworks, which service Hadoop.
2. *Distributor business model*: In this business model unique to open source, a company sells subscriptions to software and associated services that are partly or completely based on open source software. This model only works for complex software that consists of tens or hundreds and sometimes thousands of possibly incompatible components that a customer wants to use.

   The most well-known examples are Linux distributors like Red Hat and Suse, but many other smaller companies provide distributions of other kinds. The competitively differentiating intellectual properties of a distributor are its test suites, configuration databases, and compatibility matrices, which they typically do not open source.

3. *Single-vendor open source business model*: In this model, a company goes to market by providing a sometimes reduced, sometimes complete, version of its product as open source. The company never lets go of full ownership of the software and sets up various incentives for users to move from the free open source version to a paid-for, commercially licensed version. The most common incentives are support and update services, but it often also includes a copyleft license that users would like to replace with a proprietary one.

   If done correctly, both the company and its products benefit from the help of the community of nonpaying users. The company typically does not get code contributions, but it does get lively discussion forums, bug reports, feature ideas, and word-of-mouth marketing. The most well-known example of this model was MySQL, the database company, but there are many more recent ones, such as SugarCRM, MongoDB, and Redis Labs.

The distributor and single-vendor models are especially important because they enable returns on investment that are attractive to venture capitalists. Thus, they are the main conduit through which billions of dollars have been invested into open source software.

Open source foundations

There are two more models that determine how the development of open source software is being funded. They are actually two variants of the same idea: the open source foundation.

An open source foundation is a nonprofit organization tasked with governing one or more open source projects, representing them legally, and ensuring their future. In the past, open source foundations were set up to ensure the survival of unsupported community open source projects, but companies are increasingly coming together to set up a foundation with the goal of developing new open source software.

The two variants of open source foundations are as follows.

1. *Developer foundations*: This type of nonprofit foundation is run by software vendors (developers) who decide to join forces to ensure the survival and health of the open source software they depend on. By ensuring broadly shared ownership of the software, the vendors make certain that no one can monopolize this particular type of component and reap all of the profits from software products that rely on it. This is why Linux was supported against Microsoft Windows, Eclipse against Microsoft Visual Studio, and, more recently, OpenStack against Amazon Web Services.
2. *User foundations*: This type of nonprofit is predominantly run by companies that are not software vendors but rely on the software managed by the foundation, either as part of their operations or directly as part of a product that is only partly software. Examples are the Kuali Foundation for software to run universities, the GENIVI foundation for automotive infotainment software, and the openKONSEQUENZ foundation for software for the (German) smart energy grid (the last of which I helped create).

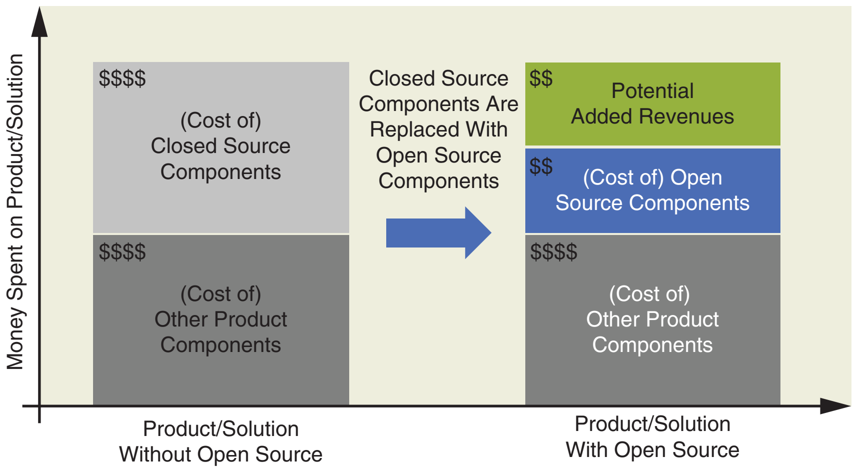
Money Spent on Product/Solution
$$$$
(Cost of) Closed Source Components
$$$$
(Cost of) Other Product Components
Closed Source Components Are Replaced With Open Source Components
$$
Potential Added Revenues
$$
(Cost of) Open Source Components
$$$$
(Cost of) Other Product Components
Product/Solution Without Open Source
Product/Solution With Open Source

FIGURE 1. The economic logic of community open source software.

Figure 1 shows how replacing a closed source component in a product with an open source component shifts profits between the different component suppliers and generally leaves more profit for the vendor, which integrates the components and sells the final product. Because of this economic logic, I expect to see more product vendors and service suppliers from outside the software industry get in on the game. They will fund the development of open source components they need, taking money out of the market for such components and moving it to places where they can more easily appropriate it. Therefore, in the future, we can expect funding for open source software development to increase by a couple of orders of magnitude. **C**

REFERENCES

1. Open Source Initiative. 2018. [Online]. Available: https://opensource.org
2. Free Software Foundation, "GNU operating system." 2018. [Online]. Available: https://www.gnu.org/philosophy/free-sw.en.html
3. C. Ebert, "Open source drives innovation," *IEEE Softw.*, vol. 24, no. 3, pp. 105–109, 2007.
4. E. Raymond, "The cathedral and the bazaar," *Knowledge, Technol. Policy*, vol. 12, no. 3, pp. 23–49, 1999.
5. The Apache Software Foundation. 2018. [Online]. Available: https://www.apache.org/foundation/how-it-works.html
6. D. Riehle, "The five stages of open source volunteering," in *Crowdsourcing*, W. Li, M. N. Huhns, W.-T. Tsai, and W. Wu, Eds. New York: Springer-Verlag, 2015, pp. 25–38.
7. J. Dinkelacker, P. K. Garg, R. Miller, and D. Nelson, "Progressive open source," in *Proc. 24th Int. Conf. Software Engineering*, 2002, pp. 177–184.
8. D. Riehle, M. Capraro, D. Kips, and L. Horn, "Inner source in platform-based product engineering," *IEEE Trans. Softw. Eng.*, vol. 42, no. 12, pp. 1162–1177, Dec. 2016.

DIRK RIEHLE is the professor for open source software at the Friedrich Alexander-University of Erlangen Nürnberg. Contact him at dirk@riehle.org.

Digital Object Identifier 10.1109/MC.2019.2906962