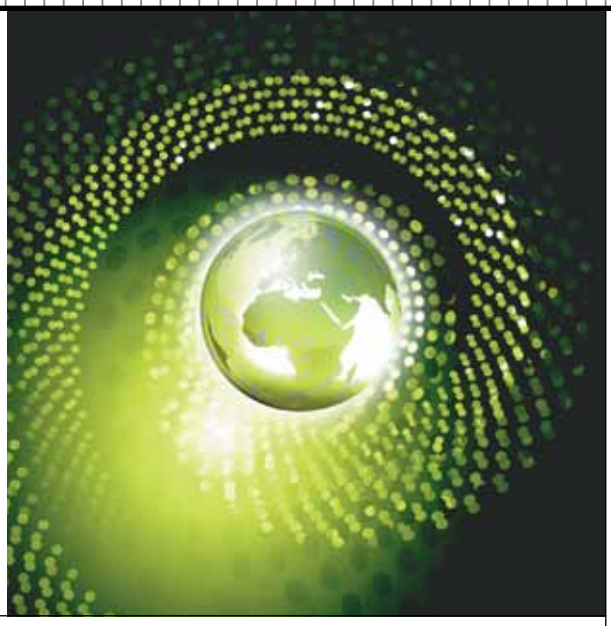
Controlling and Steering Open Source Projects



Dirk Riehle
Friedrich-Alexander University of Erlangen-Nürnberg

Commercial software firms can control or steer open source software projects to meet their business needs.

Open source software has become an important part of the software business. In a 2009 survey, Forrester Research found that 46 percent of all responding enterprises were using or implementing open source software (www.forrester.com/rb/Research/open_source_software_goes_mainstream/q/id/54205/t/2). Moreover, in 2009, the Gartner Group estimated that by 2012, at least 80 percent of all software product firms will use open source software (www.gartner.com/DisplayDocument?id=1359127).

Thus, it's important to understand how software product firms depend on open source and how they manage that dependency to meet their business goals.

There are three main types of software product firms:

- *Closed source firms* own all competitively differentiating software components their product is based on. Product examples are Microsoft's Windows, Oracle's database, and the SAP Business Suite.
- *Single-vendor open source firms* also own all competitively differentiating components, but they make some of these available as open source (D. Riehle, *The Single-Vendor Commercial Open Source Business Model*, Springer Verlag, 2011). Examples are Alfresco's same-name product, Jaspersoft's BI tool suite, and (formerly) MySQL's same-name database.
- *Open source distributors* integrate a large set of open source components and distribute the assembly for a fee. Distributors typically don't own their components. Product examples are Suse's Linux, Red Hat's Linux, and Acquia's Drupal.

What's common to these firms is that they exclusively own some intellectual property they capitalize on. In the case of closed source and single-vendor open source firms, it's the software itself; in the case of distributors, it's the branded configuration. Thus, the subject here isn't pure services firms.

It's important to understand that while open source software may be freely available under a license, it nevertheless has an owner. There are two main forms of ownership:

- *Single-vendor open source software.* This type of software has a single legal copyright owner, typically a software firm, that aggressively maintains its ownership rights.
- *Community open source software.* This type of software either has a diffuse set of owners (the code contributors) or is owned by a foundation acting on behalf of its members (D. Riehle, "The Economic Case for Open Source Foundations," *Computer*, Jan. 2010, pp. 86-90).

How then do any of the three types of software product firms depend on either of the two types of open source projects, and what are their business goals for employing open source?

SOFTWARE PRODUCTS AND OPEN SOURCE

Every real-world software product comprises multiple components, all

0018-9162/11/$26.00 © 2011 IEEE Published by the IEEE Computer Society

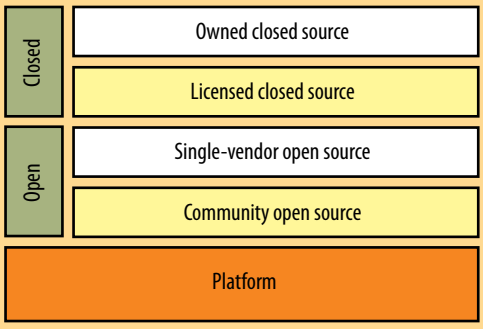


Figure 1. Software products can have four different types of components.

of which may have different licenses and hence come with different usage conditions. Figure 1 identifies the four different types of software product components.

The product illustrated in Figure 1 has both closed source and open source components. The closed source part of the stack contains components that the firm owns and licenses from other software firms to complete its product. The open source part contains single-vendor open source components that the firm owns but makes available to others under an open source license for strategic reasons as well as community open source components owned by a community of stakeholders—that is, they aren't under the control of a single legal entity.

Closed source software firms don't offer open source components; single-vendor open source firms do so by definition. All three types of software product firms use community open source. Closed source firms and single-vendor open source firms prefer community open source that comes with a permissive license to avoid the risk of having to open source their closed source components. Open source distributors use all types of open source irrespective of their license because their competitive differentiator isn't the software itself but their ability to configure and integrate the components.

An example is Jaspersoft, a single-vendor open source firm that provides business intelligence software products, most notably JasperServer, a report generator. The JasperServer enterprise edition contains closed source code that Jaspersoft owns. This version builds on the open source community edition of JasperServer, which Jaspersoft also owns. JasperServer also uses community open source packages like Hibernate, an object-to-relational database mapper. JasperServer runs on both Windows and Linux.

An analysis of various product component stacks and the corresponding firms' behavior reveals three strategic business goals:

- *Reduce development costs.* Closed source and single-vendor open source firms use community open source or license closed source to reduce development costs.
- *Maximize customer exposure.* Single-vendor open source firms actively promote their open source project with the strategic goal of selling a superior product faster at lower costs.
- *Minimize competition.* Open source invites competition, and both single-vendor open source and open source distributors employ various strategies to keep competition at bay.

To reach these goals, the software firm must actively manage or at least influence the open source projects it depends on.

CONTROL POINTS AND STEERING MECHANISMS

Software products firms use a toolbox of resources to manage open source projects and support their business goals. There are two basic types of resources: *control points* are enforceable through the legal system, while *steering mechanisms* aren't enforceable through the legal system, but are based instead on social contracts and corresponding behavior.

Control points

Software firms use several control points to gain significant control over an open source project and achieve their business goals.

Copyright practices. While a software firm can grant third parties usage rights through an open source license, it also can retain ownership of the copyright. As the owner, the firm can prevent third parties from using the software under a different license than the open source license.

Single-vendor open source firms use copyright practices to minimize competition. These firms open source some or all of their product components, giving others corresponding usage rights to their software. However, for components considered competitive differentiators, these firms want to avoid having competitors utilize their work to compete with them.

Single-vendor open source firms can maintain effective ownership by requiring outside code contributors to sign a copyright transfer agreement. To hinder the creation of competing software, they can use an aggressive reciprocal license for the open source software that requires any competitor's software built on the open source software to be open source as well.

By maintaining copyright ownership, the single-vendor open source software firm can sell a traditional commercial license while still reaping the benefits of an open source strategy.

The details of contributor agreements and open source license choice depend on the particular single-vendor open source firm's business model. We expect increased use of the Affero GNU general public license as the world moves into the cloud.

Trademark practices. Most software embeds trademarks in the form of logos, slogans, or names. Trademark owners can stop third parties from using trademarked open source software as is, requiring the potentially expensive removal of the trademarks.

Trademark practices are most important for open source distributors, but single-vendor open source firms also use them. Closed source software vendors use trademarks as well, but not in open source projects. Vendors use trademark practices to minimize competition. Open source distributors heavily invest in their brand and corresponding trademarks because that's what sets them apart from competitors who use their work.

Trademarks, like copyrights, are exclusion rights. Thus, competitors can't use a firm's distribution as is, but must first remove the distributor's trademarks. The original distributor may want to make it as hard as possible to remove its trademarks to maximize the delay between its own release and a competitor's rerelease of the same software.

In addition, removing well-known branding reduces the software's value from the customers' perspective as the original firm and its capabilities obviously no longer stand behind it. Moreover, certification programs tied to the branded distribution further increase its value while decreasing the value of nonbranded or rebranded distributions.

Domain ownership practices. Owning domains that users and customers look up for information about a product is an important means of influencing customer perception. Trademarks can prevent competitors from using domains with similar names.

Because software product vendors use domain ownership to increase customer exposure, Internet domain names are typically reflective of the firm's products. Like trademarks, this applies to all three types of firms, but only single-vendor open source and open source distributors use domain ownership to protect their open source intellectual property.

Trademark practices are most important for open source distributors, but single-vendor open source firms also use them.

Both single-vendor open source firms and open source distributors maintain domains that represent their products to interested parties. By managing the corresponding websites, they determine what third parties learn about the software, which in turn supports their respective business goals.

These firms then use trademarks to exclude competitors from buying and using domains that might infringe on their trademarks, thus maximizing customer exposure while minimizing competition.

Steering mechanisms

The control points are exclusion rights that are enforceable by law. However, while not legally enforceable, the following steering mechanisms can be equally powerful.

Social leadership practices. Open source project leaders have substantial leverage to direct that project. This includes the early license choice, the project culture and corresponding practices, and how the release plan and feature roadmap develop.

A single-vendor open source firm employs most or all of its key developers. At least some of them can fulfill public leadership roles. In this capacity, developers can nudge the community to focus their attention and possible contributions onto different aspects of the software, depending on the firm's needs.

As Eucalyptus Systems CEO (and former longtime CEO of MySQL) Marten Mickos observed, actual programming expenditures constitute only a small fraction of the software development costs—quality assurance and testing easily consume as much if not more time and money (www.parc.com/event/1092/open-for-business.html). Thus, any contribution that the firm can gather from the open source community is likely to help reduce software development costs.

In community open source, a firm can also gain influence by hiring developers and putting them to work on the project. To the extent that these developers focus on making the software work for their employer, the resulting community open source software will help the firm reduce development costs. Thus, the work of a few developers may allow for the use of a much larger software component. The more influential the employed developers, the more they'll be able to lead the community to activities that will benefit their employer and increase development cost savings.

A developer's actual influence in a community open source project evolves over time. An important factor is being a founder of the project. Most notably, 20 years after starting the Linux project, thanks to the unique hierarchical development structure that he created, Linus Torvalds remains the final arbiter of technical decisions over the Linux kernel. Other

projects—for example, the Apache Software Foundation projects—take a more egalitarian approach.

Social leadership in community open source projects can also help expand customer exposure. Highly visible developers can use their position to reach out to the community open source project's users and piggyback a message onto their outreach. That message typically conveys how well a commercial product built on it works with the community open source project they're representing. The forms of outreach range from mailing list activities to conference speaker engagements and industry publications.

Development process practices. To the extent that a firm employs the developers on a project, it can influence the development process. For example, actual development may not be public, code contributions may be time-delayed, or only snapshots may be provided.

Both single-vendor open source firms and open source distributors use development practices to minimize competition. Common practices include:

- *Closed rather than open development.* This way, competitors don't know what's coming nor can they adjust their own software to trail the changing code base.
- *Snapshots of the code base.* By not providing the full history, maintaining and working with branches becomes impossible.
- *Delayed publishing of source code after release of binaries.* The firm may choose to delay the release of the source code over the release of the binary.

The main purpose of such practices is to gain a time advantage over any possible competitors. With a significant delay before they can catch up, competitors find themselves at a disadvantage in utilizing the code. Sometimes, the firm doesn't actually have to do any of these things—the threat may be enough to discourage competitors.

Another important reason for these practices is that firms sometimes don't want their competitors to know early what they're developing. For example, hardware firms frequently perform closed development of the necessary Linux drivers for their new devices even though the Linux kernel community demands that they not do this (www.kroah.com/log/linux/stable_api_nonsense.html). Open development would let competitors learn early, from the code, about the specifics of the new devices and hence reveal the developer's competitive strategy.

Employing key developers is crucial to establishing and following an appropriate development process that fits a firm's business strategy.

Strategic positioning practices. Some marketing and outreach channels are better than others. Most notably, open source foundations provide important marketing opportunities. By locking up a project, firms can improve their customer exposure while keeping competition at bay.

Through smart strategic positioning, closed source and single-vendor open source firms can use community open source to maximize customer exposure while minimizing competition. Since they can't do it by selling the community open source, their revenue must come from an extended or complementary offering. An example is Actuate, the main developer of the BIRT report designer. Actuate's main revenue is from a complementary report generator.

Strategic positioning is best done by creating a community open source project under the auspices of an established open source foundation. The foundation lends credibility and visibility to the project. By making one of the two or more components available as community open source, closed source and single-vendor open source firms can use the supporting foundation as a marketing channel to increase customer exposure.

Software product businesses seeking to benefit from open source projects have three main goals: reduce development costs, maximize customer exposure, and minimize competition from open source. To achieve these goals, software businesses rely on a toolbox of practices using control points (enforceable by law) and steering mechanisms (social contracts). Such practices, however, need to be applied judiciously because they could lead to pushback from the open source community. A disenfranchised community might start a competing open source project that commoditizes the software vendor's offering, ultimately endangering its business.

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

Editor: Sumi Helal, Department of Computer and Information Science and Engineering, University of Florida; helal@cise.ufl.edu

Readers are encouraged to use the message board at www.computer.org/industry_perspective to post comments, offer feedback, or ask questions.