The Business of Open Source: Missing Patterns

Michael Weiss
Carleton University, Ottawa, Canada
`michael_weiss@carleton.ca`

Version 0.4 published on June 16, 2015

Abstract

This paper documents three new patterns (SUPPORT THE COMMUNITY, DUAL PRODUCT, and IP MODULARITY) that fill gaps in a pattern language for open source business published in a series of papers at EuroPLoP (Weiss, 2009, 2010; Link, 2010; Weiss, 2011; Link, 2011; Weiss & Noori, 2013). The patterns are aimed at entrepreneurs, managers in established companies, and students interested in business models that use open source.

1 Introduction

Open source has become an integral part of commercial software development. While in the past, open source software development was considered to be driven largely by volunteer effort, today most of it is carried out by companies. For example, around 70% of commits to the Linux kernel are made by for-profit companies (Linux Foundation, 2010). and about 90% of changes to the Eclipse platform are made by paid developers (Eclipse Foundation, 2015).

How companies use open source ranges from the adoption of open source development practices to the use of open source development tools; from the integration of open source components into products to active contributions to existing open source projects; and from companies initiating their own open source projects to companies collaborating on the creation of shared assets that they can jointly use to reduce the cost of product development.

This paper focuses on open source businesses, or businesses that use open source as a strategy to strengthen their business models. The paper documents patterns followed by open source businesses. Patterns are proven solutions to common problems. It complements a series of papers on open source business patterns (Weiss, 2009, 2010; Link, 2010; Weiss, 2011; Link, 2011; Weiss & Noori, 2013), adding three new patterns to the existing pattern language.

The patterns are aimed at entrepreneurs, as well as managers in established companies, who want to build a business based on open source. Students of business models that use open source should also find them of interest.

The remainder of this paper provides background on open source, patterns, and the method used to mine the patterns. It then describes the new patterns. The paper also contains an appendix with thumbnails of patterns that have been referenced in this paper, but are documented elsewhere.

2 Background

2.1 Open source

Traditional software companies made their money from selling licenses and from services such as customization and maintenance. Open source software (OSS), however, lacks the strong intellectual property (IP) protection of traditional software. Hence, companies that build their business around open source focus on selling products or services that complement the software.

The term open source owes its intellectual roots to the free software movement. The Free Software Foundation (FSF) defines free software through four rights or freedoms: the freedom to i) run, ii) study and change, iii) redistribute, and iv) distribute changes to the software. The Open Source Initiative, an industry consortium founded to create a standardized definition of open source, extends the free software definition to include the idea that open source is also a way of developing software characterized by peer review and a transparent process. To capture the notion that open source combines freedom (in the sense

Table 1: How open source strengthens a company's business model

Factor	How open source strengthens the business model
Importance	Community helps convince customers on dimensions of importance to them such as better functionality & avoiding lock-in
Stakeholder	Customers get higher quality & efficiency
value	Investors can spend more money on going to market than on R&D
	Other partners can collaborate by creating open source assets
Capabilities	Community provides company with access to talent and customers
	Puts small and large companies on equal footing
Profit formula	Allows companies to focus on IP that differentiates them
	Generates demand for paid complements

of libre, not as in free) and openness, the term Free/Libre Open Source (FLOSS) is often used. Finally, open source is no longer exclusively about software, but has been used in other combinations: open hardware, data, or science.

Open source by itself is not a business model, but a strategy that a business can use to strengthen its business model (Bailetti, 2009). We can describe a business model in terms of the importance of the problem it solves, the value it creates for its stakeholders (eg customers), the capabilities it requires to deliver that value, and its profit formula (Muegge, 2012). Table 1 enumerates some of the ways how an open source approach can strengthen a company's business model. For example, releasing an open source project helps a company generate demand for complementary products or services that it can charge for.

An open source business is a business that either i) uses open source to develop new products, ii) builds its products/services around an open source offer, iii) initiates its own open source projects, or iv) collaborates with other companies in the creation of shared open source assets. When companies leverage open source, they do so at different levels that require increasing degrees of engagement with the open source development model. The higher its degree of engagement, the higher the payoff for the company will be.

Figure 1 shows a four-level engagement model adapted from (Carbone, 2007). At the first two stages (Use and Contribute), companies leverage existing open source systems to increase their product development efficiency. At the last two

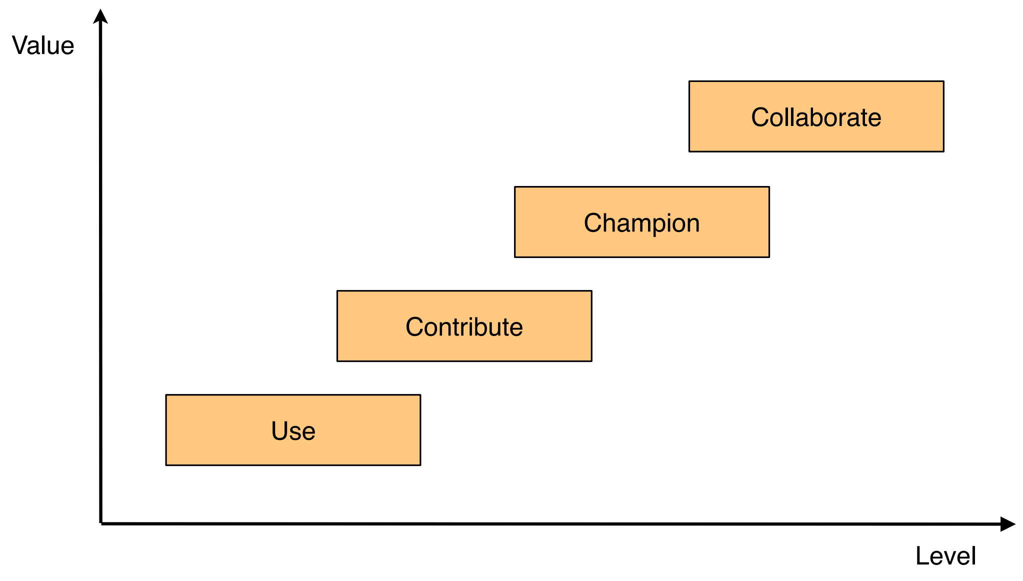


Figure 1: Open source engagement model

stages (Champion and Collaborate), they develop new ways of capturing value from products or services built around their own open source offers.

At the Use stage, companies build their products or services by incorporating existing open source components. At the Contribute stage, they also contribute back to the respective open source projects. At the Champion stage, a company initiates its own open source project, and aims to create an ecosystem around it. At the Collaborate stage, a group of companies jointly develop non-strategic assets that each member of the group can incorporate into its own projects. The Collaborate stage also involves an ecosystem, but the assets at the core of the ecosystem are now shared, unlike at the Champion stage.

2.2 Patterns

The idea of patterns stems from the work of (Alexander, Ishikawa, & Silverstein, 1977) on architecture. A pattern is a proven solution to a common problem in a given context and system of forces. Context and forces are two of the key terms in the definition of a pattern. Context describes the situation in which the pattern applies. Forces capture the conflicts within the problem.

Patterns have become popular in software design. Recently, other types of patterns have been documented for domains beyond software, eg education or business. The current paper belongs into the tradition of business patterns. An example of a business pattern is the WHOLE PRODUCT pattern (Kelly, 2012). This pattern solves a typical problem faced by a company that has made its first sales to early adopters the context. The company needs to sell more than a technology in order to appeal to mainstream customers. The solution involves selling a whole product that includes everything the customer needs.

One important fact about patterns is that they are not just clever designs, but need to reflect good practice. Pattern discovery is, therefore, also referred to as pattern mining. Until a pattern becomes recognized as a "proper" pattern, it has to undergo a rigorous process. After they have been mined by domain experts, patterns are typically submitted to a pattern conference, where they are reviewed by shepherds, and then workshopped by a group of peers.

Patterns are rarely applied in isolation. Rather, they can be thought of as words in a language. A system of related patterns is, therefore, also known as a pattern language (Alexander et al., 1977). Each pattern in a pattern language describes how it fits into a larger whole through links to other patterns. Patterns also describe the consequences of applying a pattern, ie how their solutions create benefits and liabilities; a pattern should only be applied, when its benefits are needed, and its liabilities can be tolerated or overcome by other patterns.

2.3 Method

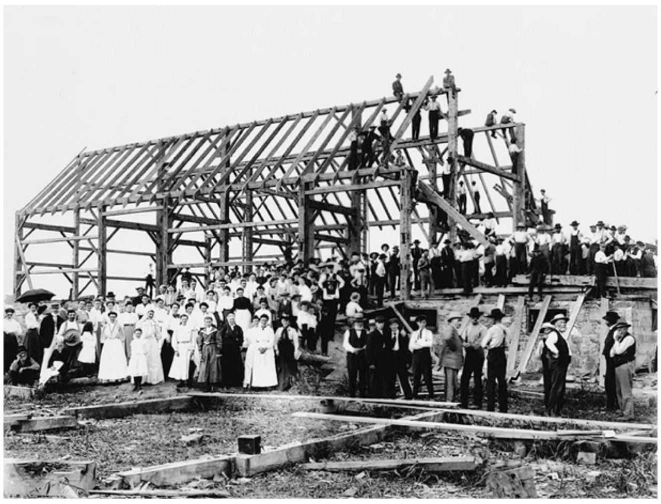
The patterns in this paper were mined from three sources: direct experience with open source projects, first-hand accounts of open source projects, and the research literature on open source development and the business of open source.

The author has been involved in several open source projects, in particular, the BigBlueButton project (`www.bigbluebutton.org`), whose goal was to create a web conferencing system. In this project, the author filled the roles of early customer (user of the web conferencing system), collaborator at the architecture and design level, and contributor of the first version of a key feature.

First-hand accounts of open source projects include those of BigBlueButton and Eclipse, a project focused on building an open software development platform (`www.eclipse.org`). Both projects are based in Ottawa, and the author had access to project leaders and members of the project ecosystems. The projects exemplify different types of settings: BigBlueButton is a project initiated by a startup, whereas Eclipse was spun out as an open source project by a large company. BigBlueButton has a small number of contributing companies at the periphery of the project (new use cases, hardware support, etc.), while the initiating company maintained hierarchical control of the project. Eclipse has a large number of contributors that range from small to large companies, and control over the project is shared among the main contributors.

These sources are complemented by a thorough review of the research literature on open source development and the business open source, covering the past 15 years and published in outlets such as Management Science, R&D Management, and Technology Innovation Management (TIM) Review, as well conferences such as the International Conference on Open Source Systems, and the Hawaii International Conference on System Sciences (HICSS).

3 Support the Community



... You have built a critical mass of functionality with → CREDIBLE PROMISE. Before contributors will join your project, there are other barriers to overcome.

* * *

You need to build a healthy community around your project.

Developers may not trust your motivation as a company (are you serious about open source, or are you just looking for free labor?).

Not only can outside contributions save you time and resources, accepting outside contributions also shows that your project is truly open.

Receiving contributions from other developers signals that the problem your project solves is relevant to others. You can test the market through an open source project.

Developers want clarity over how they can make contributions to the project and what benefits they themselves will get from contributing.

It can be daunting to contribute to a code base at an early stage, when the code is not yet well-organized and lacks documentation.

Therefore,

Support the community without expecting an immediate return. It's all about respect and building legitimacy, one step at a time.

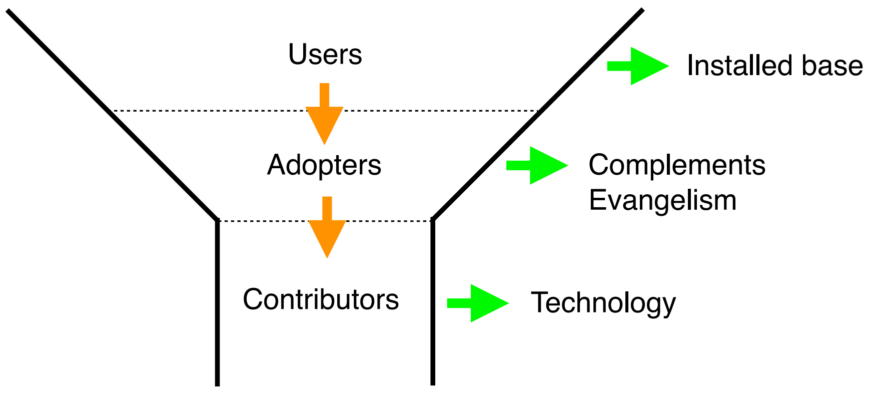


Figure 2: Community funnel

A company can build up legitimacy with the community by i) giving to the community (code, documentation, participation in the discussion forum), ii) a clear licensing practice (an open source license), iii) a clear process for making contributions, iv) making decisions in the open according to → OPEN DIALOG, and v) not treating community members as prospects (Dixon, 2011).

There will be different types of participants in your community, and you can think of them as progressing through a funnel (see Figure 2): users who use the technology internally, adopters who integrate the technology into product or services, and contributors who advance the project technology (Skerrett, 2011). They start out as users, where they constitute the installed base. They then become adopters, and create complements to and evangelize on behalf of the project. Finally, they turn into contributors to the project.

You want to cater to all three groups equally, not just to contributors. For example, if you don't actively build an installed base, you will not be able to keep the interest of your contributors or adopters. After all, one of the benefits they expect is wide-spread use of the product. Similarly, you need to spend time recruiting new adopters: their complementary products will make the your project more valuable to others. See → SELL COMPLEMENTS.

* * *

Giving contributors clear roles to get involved in the project and norms that go with those roles sets clear expectations for them.

Building a community takes time away from developing your business, but it is a long-term investment in the people who will help you succeed.

As noted in → OPEN DIALOG), openness encourages reciprocal behavior. The more support you offer your community, the more they are likely to contribute.

The more you open up your project to a community, the more the direction of the project becomes a matter of negotiation with the community. If you want more control, you should → RUN A TIGHT SHIP.

Although not specific to just this pattern, it should be restated that contributors can participate in various ways, not just in terms of code.

* * *

Reflecting on the experiences with the BigBlueButton project, Blindside Networks CEO Fred Dixon explains that "we take the perspective that, if we consciously commit a portion of our resources to assisting others on the mailing list, then as their adoption of BigBlueButton grows so does the pool of potential customers that may approach us later on" (Dixon, 2011).

The community funnel approach is used by the Eclipse project (Skerrett, 2011). Eclipse is one of the largest open source projects and consists of many vendor-specific subprojects with their associated communities. Skerrett (2011) notes that "[s]uccessful open source communities appeal to all types of participants", and that project mantainers need to "consider what best practices and strategies are needed to encourage different types of participants".

4 Dual Product

... When you → SELL COMPLEMENTS, you are not limited to selling service or support. Instead, you can play with what you release as open source.

* * *

You want to entice commercial users to pay for an open source product.

Non-paying users help drive up the installed base. The higher the installed base, the more likely you can attract adopters to → SELL COMPLEMENTS.

A free basic version of your product can be used to test the demand for your product and by customers to trial the product before buying a fuller version.

Some users may be satisfied with the capabilities of the basic version.

Commercial users have higher demands on the functionality of your product, and are willing to pay for additional features.

Non-paying users can later be upgraded to commercial users once their needs increase. The free version can be used as a "bait" for non-committed users.

Therefore,

Keep parts of the open source project closed, and sell a commercial version of the product with exclusive features.

The commercial version of your product differs from the open source version in terms of features that are not included in the open source version.

This strategy is also known as Open Core (Daffara, 2011).

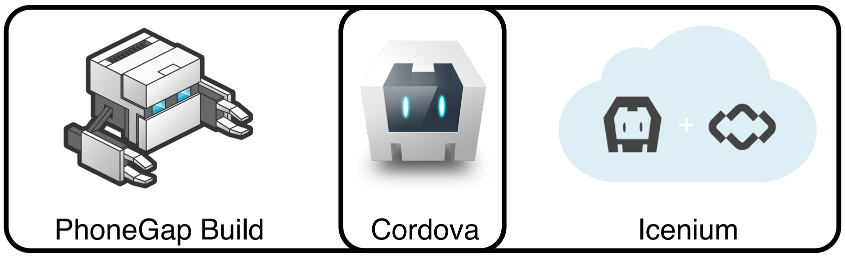


Figure 3: Two dual products based on the same open source project

* * *

This pattern presents an alternative to a → DUAL LICENSE. Rather than offering the same code under an open source and a closed source license, open source and closed source versions of a → DUAL PRODUCT differ in their features. Unlike with → DUAL LICENSE, however, you do not need to have full control over the open source product in order to create a → DUAL PRODUCT.

If the functionality of the basic version is too limited, using this pattern may backfire. It limits expectations about the quality of your for-pay version.

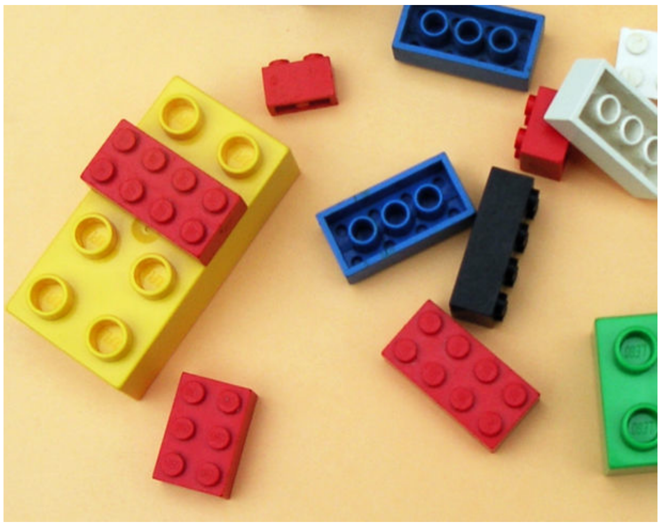
You also need to be upfront with the open source community about your approach to make money, as suggested in → SUPPORT THE COMMUNITY.

Historically, the → DUAL PRODUCT model was introduced as an afterthought, when the project owners realized that the project had commercial value, and they could sell upgraded versions of open source components. When the model is used from the start, it is easier to separate code modules according to IP.

* * *

PhoneGap, a popular development framework for mobile applications, is an Apache project led by Adobe. PhoneGap was donated by Adobe in 2011, as suggested by → DONATE CODE. PhoneGap is the brand, Apache Cordova the project. Adobe levers the PhoneGap brand to sell a cloud-based service, PhoneGap Build. A similar service is offered by Icenium. Figure 3 shows the relationship between the dual products and the open source project.

5 IP Modularity



... A → DUAL PRODUCT has open and closed components. You want to be able to generate open source and commercial versions of this system.

∗ ∗ ∗

You need to manage open and closed components of a dual product.

Components of your product represent knowledge (intellectual property or IP) that you may want to protect. This knowledge can come in different forms: copyrights, patents, or trade secrets. You may also be using ("in-sourced") components developed by other parties, which come with their own IP.

Different users may require different levels of access to your product. For example, ecosystem partners need to see code they integrate with.

You need to reconcile distributed value creation and value capture. By relinquishing control over your code base, distributed value creation leads to wider adoption and attracts external contributors, but makes it more difficult to capture value (ie make a profit) and differentiate your product.

When your product is a platform, this conflict is particularly pronounced. You want to be open to your users and complementors to enable distributed value creation, but also be able to sell your own proprietary modules.

Therefore,

Align intellectual property (IP) with product architecture.

Separate the intellectual property (IP) elements of your product by aligning IP boundaries with the technical architecture of the product (Waltl, Henkel, & Baldwin, 2012). This means that parts of the system that are licensed differently are contained in different modules within the system. The features of your product and their associated licenses can then be managed as separate architectural units. A particular configuration of the system can now be generated by extracting the respective parts from one and the same code base.

* * *

IP modularity facilitates the development of complements, which can have different licenses, and thus increases the platform's attractiveness.

However, IP boundaries may not be optimal from a technical or organizational perspective. Eg, they can reduce your performance.

It can be difficult to cleanly separate IP in the code base due to architectural choices that were made in the past without regard for IP issues.

* * *

SugarCRM is a platform for customer relationship management. As shown in Figure 4, it maintains a proprietary code tree, from which both open source and commercial configurations can be generated (Waltl et al., 2012). Reporting is an example of a module that is only included with the commercial version and released under a commercial license. Similarly, SugarCRM offers a closed source module that supports Oracle databases. Note that these are independent licensing decisions. Somebody else could develop an open source Oracle database binding without impacting the IP of the Reporting module.

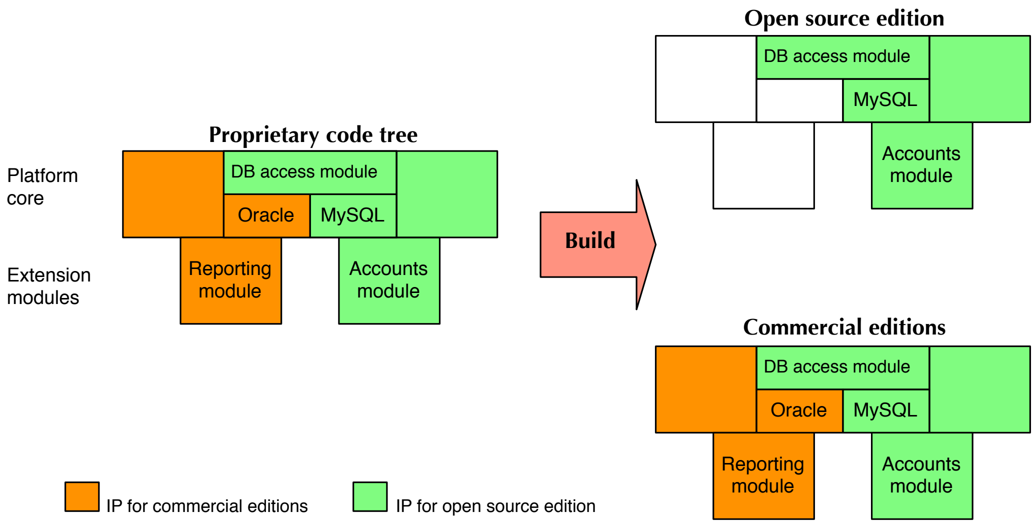


Figure 4: Example of IP modularity

6 Conclusion

In the process of writing this paper, three missing core patterns were identified: SUPPORT THE COMMUNITY, DUAL PRODUCT, and IP MODULARITY. These patterns are seen as filling essential gaps in the current pattern language.

Appendix

Thumbnails of the patterns referenced in this paper are provided in Table 2.

Acknowledgements

I want to thank Tim Wellhausen for shepherding this paper and his insightful comments. If I could not address all of his points, it's not for lack of prodding, but that he touched on some issues that suggest further exploration. I would also like to thank the participants in previous EuroPLoP workshops, where many of the patterns referenced here were first discussed, for their feedback.

Table 2: Thumbnails of referenced patterns

Pattern	**Description**
CREDIBLE PROMISE (Weiss, 2009)	To mobilize developers to contribute to your project, build a critical mass of functionality early in your project that demonstrates that the project is doable and has merit.
OPEN DIALOG (Weiss, 2009)	To engage others in your project, conduct the project in the open, maintaining a two-way dialog with project participants (users and external developers).
DONATE CODE (Weiss, 2011)	To extend the lifespan of your closed code, release it under a permissive license that allows others to build on it easily.
DUAL LICENSE (Link, 2011)	To entice commercial users to pay for an open source product, offer the same product under an open source license and under a commercial license.
RUN A TIGHT SHIP (Weiss, 2011)	To keep control of your project's direction, maintain full ownership of the code.
SELL COMPLEMENTS (Weiss, 2010)	To monetize an open source product, sell products or services (eg plug-ins, specialized hardware, or support) that complement the open source product.

Credits

Unless otherwise noted, the images were sourced from Wikimedia Commons (`http://commons.wikimedia.org`) and Flickr (`https://www.flickr.com/creativecommons`) under a CC-SA (Creative Commons ShareAlike) license.

References

Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction*. Oxford.

Bailetti, T. (2009). How open source strengthens business models. *Open Source Business Resource*. `http://timreview.ca/article/226`.

Carbone, P. (2007). Competitive open source. *Open Source Business Resource*. http://timreview.ca/article/93.

Daffara, C. (2011). Open source license selection in relation to business models. *Open Source Business Resource*. http://timreview.ca/article/416.

Dixon, F. (2011). Lessons from an open source business. *Open Source Business Resource*. http://timreview.ca/article/441.

Eclipse Foundation. (2015). *Company/project commit details*. http://dash.eclipse.org/dash/commits/web-app/commit-count-loc.php (retrieved on Jan 31, 2014).

Kelly, A. (2012). *Business Patterns for Software Developers*. Wiley.

Link, C. (2010). Patterns for the commercial use of open source: Legal and licensing aspects. *European Conference on Pattern Languages of Programs*, 7:1–7:10.

Link, C. (2011). Patterns for the commercial use of open source: License patterns. *European Conference on Pattern Languages of Programs*.

Linux Foundation. (2010). *Linux kernel development*. http://www.linuxfoundation.org/docs/lf_linux_kernel_development_2010.pdf.

Muegge, S. (2012). Business model discovery for technology entrepreneurs. *Technology Innovation Management Review*. http://timreview.ca/article/545.

Skerrett, I. (2011). Best practices in multi-vendor open source communities. *Open Source Business Resource*. http://timreview.ca/article/409.

Waltl, J., Henkel, J., & Baldwin, C. (2012). IP modularity in software ecosystems: How SugarCRM's IP and business model shape its product architecture. *International Conference on Software Business*, 94–106.

Weiss, M. (2009). Performance of open source projects. *European Conference on Pattern Languages of Programs*, A-5:1–A-5:15.

Weiss, M. (2010). Profiting from open source. *European Conference on Pattern Languages of Programs*, 5:1–5:8.

Weiss, M. (2011). Profiting even more from open source. *European Conference on Pattern Languages of Programs*, 1:1–1:7.

Weiss, M., & Noori, N. (2013). Enabling contributions in open source projects. *European Conference on Pattern Languages of Programs*, A-1:1–A-1:8.