Profiting from Open Source

MICHAEL WEISS, Carleton University

The patterns in this paper describe open source business models. Many businesses now incorporate open source, either leveraging open source to develop new products or starting their own open source projects and building their products and services around their open source offerings. The patterns in this paper aim to provide entrepreneurs, managers and students of business models with a language for creating new business models around open source, or for incorporating open source into existing business models.

Categories and Subject Descriptors: D.2.11 [**Software Architectures**] Patterns; K.6.0 [**Management of Computing and Information Systems**] Economics; D.2.9 [**Management**] Copyrights

General Terms: Design, Economics, Management

Additional Key Words and Phrases: Patterns, open source, business models

1. INTRODUCTION

The patterns in this paper describe open source business models. How companies can make money from open source is a frequently debated topic. However, since the origins of open source, our understanding of open source business models has significantly evolved. Many businesses now incorporate open source, either leveraging open source to develop new products or starting their own open source projects and building their products and services around their open source offerings. The question is not whether to use open source, but how to go about it.

Traditionally, software companies made their money through software licenses and services that complement the software including customization, training, support, and maintenance, or the sale of hardware that runs the software (eg routers or medical devices). Open source software lacks the strong intellectual property (IP) protection of traditional software. Hence, strategies for capturing value from open source software place more emphasis on complementing the software with other products or services. Licensing still plays a role, but a different one.

The audience for these patterns are entrepreneurs who seek opportunities to create new products based on open source, managers in established companies who need to compete with new market entrants that leverage open source, and students of business models. The patterns aim to provide them with a language for creating new business models around open source, or for incorporating open source into existing business models.

2. OVERVIEW OF THE PATTERNS

A map of the patterns showing their relationships is shown in Figure 1. Links between patterns X and Y should be interpreted as “after pattern X you may also use pattern Y”. A paragraph symbol (§) after a pattern name means that this pattern is (will be) described elsewhere. Thumbnails of these patterns can be found in the Appendix.

There are different ways of leveraging open source for profit. The most direct is to use open source in product development. Companies can BOOTSTRAP their product development by building on existing open source, rather

Author’s address: M. Weiss, Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Dr, Ottawa, ON K1S 5B6, Canada; email: weiss@sce.carleton.ca

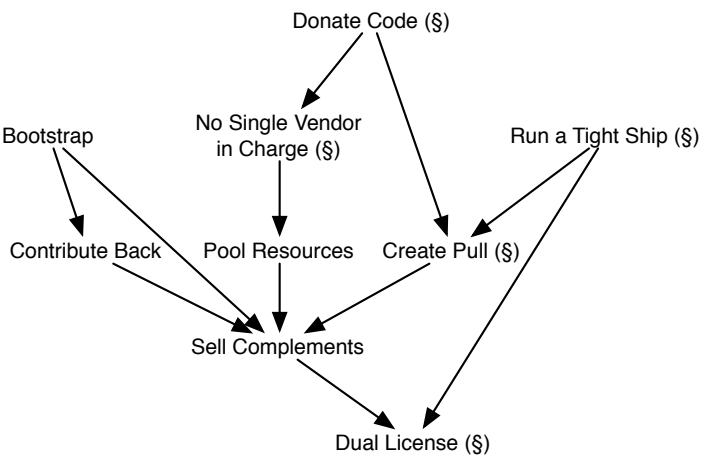

Fig. 1. Patterns for profiting from open source.

than developing a system from scratch.[1] This also allows you to increase your share of the customer's wallet as more of your effort is spent on adding value that the customer is willing to pay for.

A more powerful lever is to CONTRIBUTE BACK to the open source project that you leverage when you BOOTSTRAP your development effort. By contributing back to the projects you leverage you keep your project aligned with the evolution of those projects, and you help ensure their sustainability.

Both of these levers help you reduce cost (development and maintenance), and speed up your development. A way to generate revenue is to SELL COMPLEMENTS to an open source product, a strategy that can be applied whether or not you are the creator of the product. While the open source product itself is difficult to monetize, you can charge for products or services that complement it (such as support or specialized hardware).

When you create your own open source project, a key decision is whether or not you should RUN A TIGHT SHIP (§) by keeping control of the direction of the open source project. Typically, you would contribute most of the resources for developing the software in this case. However, there is a benefit: as owner of the code you can DUAL LICENSE (§) it, offering a free and a for-charge version, which is a special case of SELL COMPLEMENTS.

A company can also DONATE CODE (§) to open source that was proprietary but no longer provides you with a competitive advantage. This can be used to extend the lifespan of your software by giving others a platform to use, and to create demand for complementary products and services that you offer.

Contributing to open source is a way of reducing your marketing expenses. In a typical software-based product only a small portion of the cost is directly related to development. Allowing customers to download your software for free CREATES PULL (§) for your product and its complements.

To gain wide-spread adoption of code that you donate and get other companies (including your competitors) to contribute you need to give up ownership over the code by setting up a foundation and assigning ownership of the code to it. NO SINGLE VENDOR IN CHARGE (§) builds trust and facilitates collaboration among different companies.

If there is NO SINGLE VENDOR IN CHARGE (§), companies can POOL RESOURCES with other companies who can all benefit from the jointly created software. Every company that contributes to the pool can leverage the pooled resources to SELL COMPLEMENTS through which they differentiate themselves.

[1] When referring to a pattern, this paper puts the pattern name in small caps.

3. BOOTSTRAP

> *In 1999, Jason Fried and partners started 37signals, a successful provider of project management and collaboration software, such as Basecamp, on a shoestring budget of $150. Says Fried, "It costs virtually nothing to start a software business these days." Cheap open source web tools reduced the development and marketing costs for 37signals.* [Fitzgerald 2006]

3.1 Context

You are developing a new software-based product.

3.2 Problem

How do you keep the costs for developing your product low?

3.3 Forces

Developing software from the ground up gives you great flexibility, but is time-consuming and expensive.

Leveraging existing work allows you to focus your development effort on the value added by your product, which is how you can differentiate yourself from competing products.

3.4 Solution

Bootstrap product development by reusing existing open source components in your products to get something working quickly and keep costs low.

The notion of bootstrapping captures the idea of starting a business on limited resources [Kawasaki 2004]. A bootstrap is the loop on some boots by which to pull on the boot. A company that bootstraps itself cannot afford to spend time on activities that do not add value to customers, and help it generate revenue. When you build your product on top of open source components, you will more quickly get to a point where you start adding real value for your csutomers. You will also keep yours costs low in building that first version.

Building on open source can also increase your share of the customer's wallet [Riehle 2010]. Assuming that customers have the same budget to spend, more of their money will go towards functionality that add value, if you leverage open source. The reason for this is that using open source reduces your cost in developing the base functionality, and increases your bottom line. You can even pass on some of the savings for a mutual win-win: more revenue to you, and more real value to your customers.

3.5 Consequences

Reusing open source components allows you to develop functionality quickly at low cost. The more you can focus on real value for your customers, the better for you and them.

Over time, you may find that building on many open source components you also inherit functionality that you never exploit. Your codebase will be larger than a custom-built solution. The code will also be affected by changes to the components you use, and you need to align your product with those changes.

3.6 Examples

The initial version of the BigBlueButton open source web conferencing system [BigBlueButton 2010] was built by combining 14 different existing open source components, ranging from Red5, a streaming server for managing voice and video streams, to Open Office for converting slides from various formats. This approach kept the cost of developing the system low and sped up the creation of the first version.

The Maemo platform for Nokia's Internet tablet incoporates many open source components [Stuermer et al. 2009]. According to Stuermer et al. [2009], about 25% of the code are unmodified open source components. An additional 50% incorporate changes and improvements made by Nokia, and only 25% is closed code that Nokia developed itself or with help of contractors. The closed code is related to the unique user interface of the tablet

and the hardware layer. Using open source, Nokia shortened its time-to-market. However, it also proved difficult to integrate many open source components, because Nokia lacked the knowledge. Nokia mitigated some of those challenges by hiring contractors, who were familiar with those components.

3.7 Related patterns

To simplify the bootstrap process required to produce a critical mass of functionality, BUILD ON THE SHOULDERS OF OTHERS (§). Reusing components with proven functionality also allows a project to reach a higher level of quality earlier (ie through reuse the project will improve in terms of quality and time to market).

Pay attention to the component licenses, so PLAY BY THE RULES (§) those licenses impose. You may need to rewrite code that you used during initial development when you deploy the product.

3.8 Sources

Literature on reuse in open source [Haefliger et al. 2008], on the Maemo platform Stuermer et al. [2009], on commercial open source Riehle [2010], and the author's observation.

4. CONTRIBUTE BACK

> *Getting more done for less is the obvious benefit of open sourcing your work. But there are a lot of other positives as well. For one, it feels good to give back. 37signals as a company is possible in part only because of how open source lowered the barriers of entry for small businesses. [...] Everything from the operating system to the database to the web server to the proxy engine [...] is open source. Helping further that ecosystem is very rewarding.* [Heinemeier Hansson 2007]

4.1 Context

You are building your product on top of an existing open source project as suggested by BOOTSTRAP. From a business perspective, you need to ensure that the project remains alive, and that you are able to incorporate future changes into your product.

4.2 Product

How do you keep aligned with the open source project?

4.3 Forces

Changes to the project require you to incorporate those changes back into your version of the source code.

The open source project may evolve in a different direction from where you would like it to go.

Such changes may be difficult to understand.

Other members may not trust your intentions when you join.

Keeping adaptations to the project secret may allow you to maintain a competitive advantage.

4.4 Solution

Contribute non-core modifications back to the open source project.

When you make changes to the code of a project that are specific to your context, but are not core to your product, you should contribute those changes back to the project. It reduces the effort that you would otherwise spend on synchronizing the changes to the project by others with modifications you made to the code, and you can focus your resources on the activities that add value to your product. By supporting the project you also ensure that it stays alive. In any case, if the project has a reciprocal license, you are required to contribute back modifications, if you distribute them in a product, although you do not have to do so otherwise.

4.5 Consequences

Participating in a project allows you to monitor its evolution, so you become aware of changes.

When you actively contribute to the project you can influence its direction.

Contributing to the project provides you with the skills to understand (absorb) changes and leverage them.

You also gain the trust of the existing project members.

Contributing to an open source project may reveal information about future products to your competitors.

4.6 Examples

Companies like IBM and Sun all contribute to the development of the Mozialla Firefox browser.. Their contributions consist of code that makes Firefox compatible with their platforms. See also SELL COMPLEMENTS for another perspective of this example.

4.7 Related patterns

OPEN DIALOG (§) helps you engage other project participants.

COPYLEFT (§) and PLAY BY THE RULES (§) may affect how you need to license your contributions. For example, if the project uses a share alike license, you need to license your code in a similar way.

4.8 Sources

Literature [Dahlander 2006], and the author's observation.

5. SELL COMPLEMENTS

> *Sun and HP are hardware companies. They make boxen. In order to make money on the desktop, they need for windowing systems, which are a complement of desktop computers, to be a commodity. Why don't they take the money they're paying Ximian and use it to develop a proprietary windowing system? They tried this [...], but these are really hardware companies at heart with pretty crude software skills, and they need windowing systems to be a cheap commodity, not a proprietary advantage which they have to pay for.* [Spolsky 2002]

5.1 Context

You are already reducing cost (development and maintenance), and shortening your development time using BOOTSTRAP and CONTRIBUTE BACK. In order to grow you need to generate revenue, not save cost.

You CREATE PULL (§) to gain adoption of your open source product, and POOL RESOURCES to maximize the return on your effort.

5.2 Product

How do you monetize an open source product?

This question arises whether you are the creator of an open source product, or whether you want to monetize another project.

5.3 Forces

An open source product is difficult to monetize directly.

It is difficult for a single company to satisfies all customer needs.

Different groups of customers have different needs.

5.4 Solution

Sell products or services (such as hardware or support) that complement the open source product.

Companies following this pattern offer more stable or enhanced versions of their product to paying customers.

There are various ways to complement an open source project: creating different versions of the product (eg a community and an enterprise edition), packaging and distribution, providing technical support, training or customizations, or building hardware to run the product. Companies can charge for complements.

When the complements are software add-ons that extend the base open source product, a strategic choice is to decide which add-ons are part of the open source product and which ones you want to charge for.

You can further increase the adoption of your product by allowing THIRD PARTIES (§) to develop and sell their own complements, rather than providing all of the complements yourself.

5.5 Consequences

You can generate revenue by charging for complements that increase the value of your open source product.

Other companies (complementors) can create complements for your product, but you need to share the value created by your product with them.

You can bundle your core open source product in different ways (different configurations) with complements to meet the needs of different customers.

5.6 Examples

RedHat simplifies the task of selecting and installing the Linux operating system by creating and selling a distribution. A distribution is a collection of open source components that have been certified to work well together. RedHat makes money by providing regular updates to their distribution through a subscription. They also offer paid support, which is attractive to companies who would hesitate to use unsupported software.

The creators of the BigBlueButton web conferencing system sell add-on modules such as desktop sharing to business users. These add-ons meet the needs of business users of the system who are willing to pay for them. Some add-ons are developed and paid for as contract work for a specific customer, others are designed for a customer segment and offered to them for licensing. More price-conscious users such as educational institutions will prefer to use the base version of the product or develop their own add-ons.

IBM is a strong supporter of Linux. It has made Linux the operating system of choice for its servers, and has ported most of its applications so they would run on Linux. IBM's support for Linux drives demand for its server hardware. The server hardware complements Linux. Another example of this strategy is Sun who open sourced its Solaris operating system in 2005, and sells servers optimized to run it. Sun's example is a combination of this pattern with DONATE CODE (§) in order to extend the lifespan of Solaris.

5.7 Related patterns

A specific use of this pattern that applies to companies who own the open source code, is to DUAL LICENSE (§) the code by providing two versions of it: a free, unsupported one with a restricted license, and one that is supported and has no license restrictions.

WHOLE PRODUCT [Kelly 2008] suggests to augment a core product with additional products and services needed by the customer.

5.8 Sources

Literature [Fosfuri et al. 2008; West and Gallagher 2006], and the author's observation.

6. POOL RESOURCES

> *[The Linux kernel development team adds] 11,000 lines, remove[s] 5,500 lines, and modif[ies] 2,200 lines [of code] every single day. [...] It's something that no one company can keep up with. It would actually be impossible at this point to create an operating system to compete against us. You can't sustain that rate of change on your own.* Kroah-Hartman, cited in [Asay 2009]

6.1 Context

There is no SINGLE VENDOR IN CHARGE (§) of the open source project.

6.2 Product

How do you maximize the use of your resources?

6.3 Forces

Companies spend the majority of their development resources on developing assets that do not add value.

These assets cannot be monetized directly, but it can be expensive to build them or involve investments that cannot be recovered; eg if a company builds software that implements a particular standard they cannot reuse that software, if the standard is not adopted.

Non-contributors can also benefit from the developed assets.

6.4 Solution

Pool resources with other companies to jointly develop a common stack of open source assets that the companies can all build on to develop their individual products.

6.5 Consequences

Companies can focus on the added value that differentiates them from others.

Cost and risk of building the shared assets can be spread across all the companies that contribute to the project.

Contributors know how to use the assets effectively; non-contributors who just use the common assets without helping create them (these are also known as “free-riders”) have not gone through the same learning.

6.6 Examples

Companies like IBM and Sun all contribute to the development of the Mozilla Firefox browser so it would run on their systems. The core business of these companies is selling workstations. Browsers are not a differentiating attribute of their offer, so they share the development costs, requiring only an investement in customizing the shared browser technology to their systems.

The members of the Eclipse project develop common assets (such as GUI components and code generation tools) that each of them requires, but that, on their own, do not create value for their customers. On the other hand, developing such assets requires them to dedicate resources. All members win by sharing those development costs with other members, and concentrating on areas in which they can differentiate their offers from each other.

6.7 Related patterns

Companies can leverage the jointly developed assets to sell complements that add value to their customers.

6.8 Sources

Literature [West and Gallagher 2006], and the author’s observation.

7. ACKNOWLEDGEMENTS

I thank Allan Kelly for offering to shepherd this paper and his wealth of insight. If I haven’t been able to incorporate all of his valuable suggestions, it is simply that he puts the bar very high and I very much appreciate that. I would also like to thank the EuroPLoP 2010 workshop participants for their helpful feedback.

APPENDIX

Here are short forms of the patterns not described in this paper:

Build on the Shoulders of Others	Integrate assets from other open source projects to grow a critical mass of functionality [Weiss 2009].
Copyleft	Constrain others to propagate the license attached to your code, if you want to tightly control its use.
Create Pull	Make it easy to access your product and leverage distributors, partners, and community members to market your product.
Donate Code	Release proprietary code under a flexible license that allows others to build on to extend its lifespan.
No Single Vendor in Charge	Transfer ownership of the code to an independent foundation in order to attract other companies to contribute to an open source project that you created.
Open Dialog	To engage others maintain a two-way dialog with project participants [Weiss 2009].
Permissive License	Allow others to use code without license restrictions, if you want you code to be widely used commercially.
Play by the Rules	Adhere to the license terms imposed by the open source components you use to ensure compliance. This pattern is also known as respect the license [Link 2010].
Run a Tight Ship	Maintain full ownership of the code, if you want to keep control of the direction of your project.
Third Parties	Encourage others to create and sell complements.

REFERENCES

ASAY, M. 2009. At its best, is open source unbeatable?. *CNET*, Nov 25, http://news.cnet.com/8301-13505_3-10405324-16.html.

BIGBLUEBUTTON 2010. code.google.com/p/bigbluebutton.

DAHLANDER, L. 2006. A man on the inside: Unlocking communities as complementary assets. *Research Policy*, *35(8)*, 1243-1259.

FITZGERALD, M. 2006. Bootstrapping 101: Use cheap web tools. *Inc.*, www.inc.com/magazine/20060701/bootstrapping-l5.html.

FOGEL, K. 2006. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly.

FOSFURI, A., GIARRATANA, M.S. AND LUZZI, A. 2008. The penguin has entered the building: The commercialization of open source software products. *Organization Science*, *19(2)*, 292-305.

HAEFLIGER, S., VON KROGH, G., AND SPAETH, S. 2008. Code reuse in open source software. *Management Science*, *54(1)*, 180-193.

HEINEMEIER HANSSON, D. 2006. Ask 37signals: How has open source helped or hindered? *Signals vs. Noise*, Nov 7, 2007, 37signals.com/svn.

KAWASAKI, G. 2004. *The Art of the Start*. Portfolio.

KELLY, A. 2008. Business patterns for product development. *EuroPLoP*.

LINK, C. 2010. Patterns for the commercial use of open source. *EuroPLoP*.

RIEHLE, D. 2010. The economic case for open source foundations. *Computer*. IEEE, Jan 2010, 86-90.

SPOLSKY, J. 2002. Strategy Letter V. http://www.joelonsoftware.com/articles/StrategyLetterV.html.

STUERMER, M., SPAETH, S., AND VON KROGH, G. 2009. Extending private-collective innovation: A case study. *R&D Management*, *39(2)*, 170-191.

WEISS, M. 2009. Performance of Open Source Projects *EuroPLoP*.

WEST, J., and GALLAGHER, S. 2006. Challenges of open innovation: The paradox of firm investment in open-source software. *R&D Management*, *36(3)*, 319-331.