Architecture as Enabler of Open Source Project Contributions[1]

Michael Weiss and Nadia Noori

Carleton University, Ottawa, Canada
weiss@sce.carleton.ca

Version 0.6

1 Introduction

The patterns in this paper are part of a larger pattern language for open source businesses. Other parts of the pattern language (Weiss, 2009; 2010; 2011) discuss the performance of open source projects and open source business models. The focus of these patterns is on the role of architecture in getting external developers to contribute to open source projects.

Architecture affects the ease with which open source components can be integrated; how contributions to an open source project can be made; what parts of the product are under the control of the project initiator or champion, and how straightforward it is to create different configurations of the product; and how collaboration among contributors is organized.

The audience of this paper includes companies starting open source projects. It also includes potential contributors to open source projects, and anyone interested in how open source projects work. The remainder of this paper provides a narrative that connects the patterns and puts them into perspective of earlier and future patterns; it outlines how the patterns were discovered; and it presents two architecture patterns in detail.

2 Patterns

To attract contributors to your open source project you need to create conditions that are attractive to contribution. First, you need to create a CREDIBLE PROMISE (Weiss, 2009). Contributors must find it worthwhile contributing to your project. To that end, the project needs to offer them something that they either could not create on their own, or only at higher cost. You must also allow contributors to work on important tasks that challenge contributors and allow them to create value for themselves, not just peripheral ones of little complexity and value.

Second, you need to follow a development process that emphasizes openness and sharing – see FREQUENT RELEASES and OPEN DIALOG in Weiss (2009). However, even when a minimal codebase and development process are in place, contributors might find it difficult to contribute to your project. The third enabler of contribution is a MODULAR ARCHITECTURE (this paper). It allows developers to contribute without requiring them to have deep knowledge of the codebase. By breaking the code into largely independent modules, the pieces of the code can be created independently, but will work together.

1 Licensed under a Creative Commons BY-SA license.

If you are starting with an existing system, you may need to RESTRUCTURE (future paper) the codebase. This happens if the code is too monolithic and hard to extend by developers who are not deeply familiar with it. You can make the code more modular by reorganizing it into subsystems with minimal interaction between them. Creating a modular architecture also requires you to commit to INTERFACES (future paper) through which the modules will interact. Contributors will write their code to match those interfaces. This ensures architectural consistency of the contributions.

A modular architecture divides the codebase into a STABLE CORE (future) and flexible complements. By keeping the components at the core of the codebase stable, you allow the components that use this codebase to vary. A guiding objective is to CREATE OPTIONS (future) for external contributors. External contributions tend to be highly specific. They are often designed to meet the needs of particular users, and it is not cost-effective to support them in the core of your codebase, nor can you anticipate such specific uses.

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". Patterns in italics will be described in future papers. Thumbnails of these patterns can be found in the Appendix.

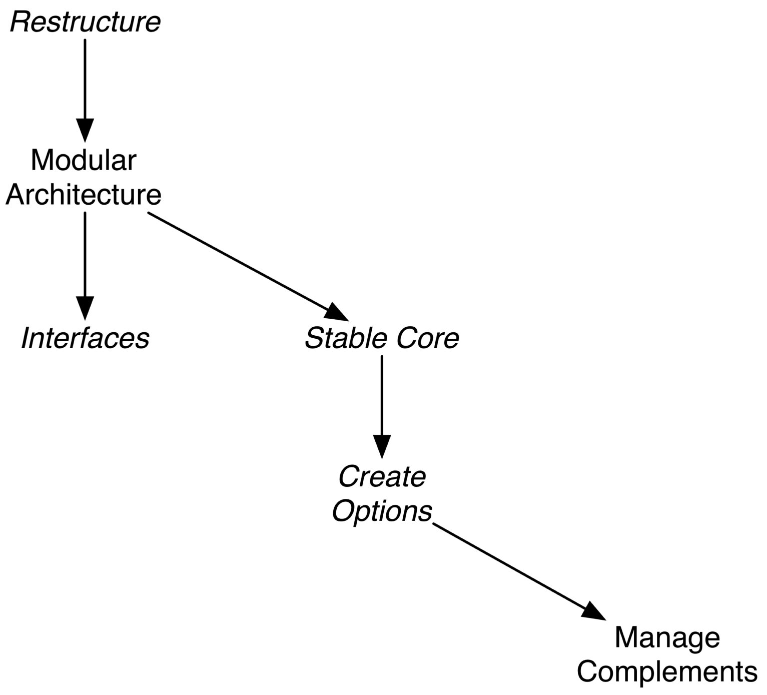
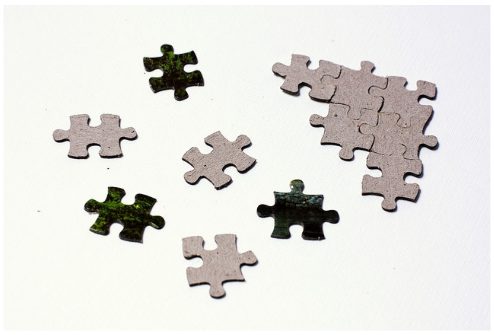


Figure 1: Open source architecture patterns

The patterns were mined from the literature, from case studies, and the authors' own experience in contributing to open source projects. The salient references on modularity in open source are Baldwin & Clark (2006) and MacCormack et al. (2006). The case studies are described in Noori & Weiss (2012; 2013). This research examined 12 open source platforms (including OpenOffice, Eclipse, and Firefox), and identified three underlying governance models, and a set of regulatory instruments (such as pricing and processes) used by the platform owners to ensure the quality of the complements. Both authors contributed to the BigBlueButton open source web conferencing system as advisors and developers.

2.1 Modular Architecture



"GNU/Linux is known for being a modular complex artifact and its successful development, accomplished by a distributed community of hackers, largely benefited from that. Therefore, it may be surprising that its core component, the so called kernel, was initially conceived as a highly integrated product and that eventually acquired a modular structure."

Rossi et al., 2003

Context You started an open source project and want to open it up to external contributors.

Problem **You want to allow external contributors to add to the codebase without requiring them to have deep knowledge of it.**

Forces Architecture plays a crucial role on leveraging the contributions of third-party developers and enabling them to add new code and functions to a project. Also, the architecture enables project owners to control project growth and helps them monitor development activities in the project network (Weiss & Noori, 2012).

Developers in open source projects need to collaborate virtually and work on the code in parallel. With a traditional monolithic architecture, it would be impossible for these virtual teams to work together because of interdependencies in the codebase that adds complexity not only to the code but to the relationships between the virtual teams that would be reflected on the community itself.

Performance is generally better, if you tightly integrate the functionality of your system. However, such tight coupling makes it more difficult to understand and extend the codebase.

Therefore,

Solution **Partition the codebase so that different parts (or modules) can be worked on and managed independently.**

Manage the complexity of expanding the codebase by limiting the scope of interaction between components. This requires you to reduce the amount and range of dependencies in your code. Often systems are not completely decomposable, however. Therefore, your goal has to be to minimize dependencies.

To reduce interdependencies map the dependencies between modules and then restructure the code. A module with a large number of dependencies is a candidate for restructuring, as any changes to such a module can impact many other modules.

Design uncertainty is also a good indicator of where you want to create modules. If you do not know how to implement a certain functionality or there are different alternatives to implement it, it is often a good idea to to create a module for this functionality. If new knowledge later reduces the uncertainty, then it will be relatively simple to incorporate the new solution without needing to change the rest of the system (Baldwin & Clark, 2000). A good example is when you deal with a quickly changing technology base and you want to make your system resilient to changes in technology.

Consequences A modular architecture defines how contributors can participate in the open source project. In the words of MacCormack et al. (2006), it creates an "architecture for participation". The separate modules in a modular architecture can be worked on independently.

A modular architecture makes the codebase extensible by dividing it into a stable core and a set of flexible complements. Complements are modules that extend the functionality of the core.[2] They are typically created by third parties. Not all complements are required for the system to work, but they can be added as desired.

Separating a system into modules can also reduce performance. Thus, new products often start with a tightly integrated architecture that evolves into a more modular architecture over time, as the ability to substitute components for one another and to distribute work outweigh reduced performance.

Known uses Examples of systems using a modular architecture are OSGi (Open Service Gateway initiative) based runtime frameworks like Apache Felix and Equinox OSGi, as well as Eclipse, an integrated development environment and rich client platform.

The Linux kernel provides stable interfaces that extensions that add support for new hardware or file systems can plug into. The Linux kernel is the core, and kernel extensions are the complements.

Many well-known open source projects such as Mozilla (MacCormack, 2006) and Apache Tomcat (Milev et al., 2009) went through rounds of restructuring, in which initial functionality was delivered in the form of an integrated architecture, and the system was was subsequently restructured to increase its modularity.

Related patterns` Creating a modular architecture also requires you to commit to INTERFACES through which the modules will interact.

A modular architecture also lays the foundation for an extensible codebase. As describe in STABLE CORE, a modular architecture divides the codebase into a stable core and flexible complements.

A high degree of modularity enables the collaborative creation of a common stack of open source components. The term "ecosystem" is often used to describe this form of collaboration between companies. These companies POOL THEIR RESOURCES (eg skills) to create common assets, and then develop their individual products by building on this common base (Weiss, 2010).

Sources Literature (Noori & Weiss, 2012; Baldwin & Clark, 2006; MacCormack et al., 2006) and the authors' observation.

2 The term "complement" originates in the platform literature (Baldwin & Clark, 2006). It formally refers to a product that adds value to the platform. A good example are smartphones (platform) and its apps (complements). There would be little value in a smartphone player without apps.

2.2 Manage Complements

"One perceived risk of using open source software components in commercial systems is open source project sustainability. It would be expensive for the project supporting a critical open source component to fail midway through the life cycle of a commercial product."

Sethanandha et al., 2010

Context You created a modular architecture and have been successful attracting third parties who provide complements.

Problem **You need to ensure the quality of third-party complements.**

Forces The quality of the complements is going to reflect on the quality of the open source product. Consider OpenOffice. The core product can be extended through extensions (eg dictionaries or themes). A poorly implemented extension may give users the impression that OpenOffice itself is of low quality (Noori & Weiss, 2012).

Managing complements come with trade-offs between the quality of contributions and the level of innovation or novelty within the open source project community (Noori & Weiss, 2013). A project needs to balance between level of contributions and their quality.

Therefore,

Solution **Establish a governance model and provide regulatory tools and processes to manage complement development.**

There can be different tiers (or circles) of complements. An inner circle contains complements that are frequently shipped with the STABLE CORE and that are highly visible to users. An outer circle comprised of complements that are either more experimental or specialized do not require the same degree of oversight.

Toolkits, development frameworks and coding conventions provide developers with guidelines for developing quality complements.

Sandboxes play another role in managing complements. They provide a controlled medium in which developers of complements can test their modules or add-ons before publishing them as part of the STABLE CORE (Venugopalan, 2005).

Consequences Some might argue that setting too many rules create obstacles and can hinder the growth of the open source project, because they might limit contributions that are more innovative and do not pass the strict “quality gates” imposed by the platform rules.

Enforcing rules and restrictions on complements ensures that the platform does not suffer damages to how it is perceived by users or to its architectural integrity due to low-quality complements.

It is important to note that managing external contributors also comes at some cost. Overseeing the contributions from external contributors creates overhead that is not related to your work on the core codebase, and it may also be difficult to convince other internal parties to allow external contributors to participate.

Known uses Eclipse has created a three-tier community (core, internal, and external), where external complements can be offered by anyone, but new Eclipse core projects (internal complements) need to be qualified. Core projects have to follow a thorough incubation process with “gates” at which their quality is assessed. Apache, OpenOffice, and SpringSource are set up similarly.

In the Apache project, if developers wish to contribute a module that extends the core of the Apache HTTP server platform, then they have to follow a well defined process. The Apache incubation process covers the establishment of a candidate, acceptance (or rejection) of a candidate leading to the potential establishment of a “podling” and associated incubation process, which ultimately leads to the establishment of a new Apache Top-Level-Project (TLP) or sub-project within an existing Apache Project.[3]

Another example is OpenOffice.org, which has established a quality assurance procedure for testing new feature components, where a quality assurance engineer is assigned to oversee the development and test processes of the new feature components.

The pattern is also used in the design of closed source software. Eg the functionality of Adobe Photoshop can be extended by third-party plugins, and there is an approval process for plugins, before they are listed on the Adobe Photoshop Marketplace.

Related paterns A company can always RUN A TIGHT SHIP (Weiss, 2010). In this case, it will maintain full control over the project. However, external contributions will be minimal, and the bulk of the contributions have to be made by the company itself.

Sources Literature (Noori & Weiss 2012; Noll 2007) and the authors' observation.

3 http://incubator.apache.org/incubation/Incubation_Policy.html

Acknowledgements

I thank Michael John for providing many helpful comments during the shepherding process. I would also like to acknowledge the members of the writers' workshop. The lively discussion gave me valuable input.

In formatting these patterns I owe a tremendous amount to the format Allan Kelly has used in his own papers, which I tried to emulate.

Appendix – Pattern thumbnails

Here are short forms of the patterns referenced in this paper.

CREATE OPTIONS (future paper)	You want developers to contribute voluntarily. Therefore, allow developers to create value for themselves that they could not create if they did not contribute.
STABLE CORE (future paper)	You want to support a variety of ways in which developers can extend the code. Therefore, keep the components at the core of the code stable, and allow the components at its periphery to vary.
INTERFACES (future paper)	You need to ensure that changes made by external developers are consistent with the architecture of the code. Therefore, document the interactions between the modules through interfaces.
RESTRUCTURE (future paper)	Your code is too monolithic and hard to extend by developers who are not deeply familiar with it. Therefore, reorganize the code to make it more modular.

References

Baldwin, C., & Clark, K. (2006), The architecture of participation: Does code architecture mitigate free riding in the open source development model?, Management Science, 52(7), 1116-1127.

MacCormack, A., Rusnak, J., & Baldwin, C. (2006), Exploring the structure of complex software designs: an empirical study of open source and propritary code, Management Science, 52(7), 1015-1030.

Milev, R., Muegge, S., & Weiss, M. (2009), Design evolution of an open source project using an improved modularity metric, Open Source Ecosysems: Diverse Communities Interacting, 20-33.

Noll, J. (2007). Innovation in open source software development: A tale of two features. In J. Feller, B. Fitzgerald, W. Scacchi, A. Sillitti (Eds.), Open source development, adoption and innovation (109-120), IFIP/Springer.

Noori, N., & Weiss, M. (2012), Managing the quality of platform complements: The case of extensions in open source software platforms, ISPIM.

Noori N, & Weiss, M. (2013), Going open, does it mean giving away control? Technology Innovation Management. (http://timreview.ca/article/647)

Rossi, A. and Narduzzo, A., (2003), Modularity in Action. GNU/Linux and free/Open source software development model unleashed, Quaderni DISA, Vol 78, Department of Computer and Management Sciences, University of Trento, http://EconPapers.repec.org/RePEc:trt:disatr:078.

Sethanandha, B.D.; Massey, B.; Jones, W., Managing open source contribuions for software project sustainability, Technology Management for Global Economic Growth (PICMET), 2010 Proceedings of PICMET '10:, vol., no., pp.1,9, 18-22 July 2010.

Venugopalan, V. (2005). Chapter 4: Developers sandbox. In CVS best pracices. Retrieved on July16, 2009, from, http://tldp.org/REF/CVS-BestPractces/html/section1-devsandbox.html.

Weiss, M. (2009), Performance of open source projects, EuroPLoP, CEUR, 566.

Weiss, M. (2010), Profiting from open source, EuroPLoP, ACM.

Weiss, M. (2011), Profiting even more from open source, EuroPLoP, ACM.

Photo credits

Puzzle pieces, shared under a CC-BY 2.0 license by Horia Varlan
https://www.flickr.com/photos/horiavarlan/4273913228

Juggler, shared under a CC-BY 2.0 license by Clan Tynker
https://www.flickr.com/photos/larry1732/6158766205