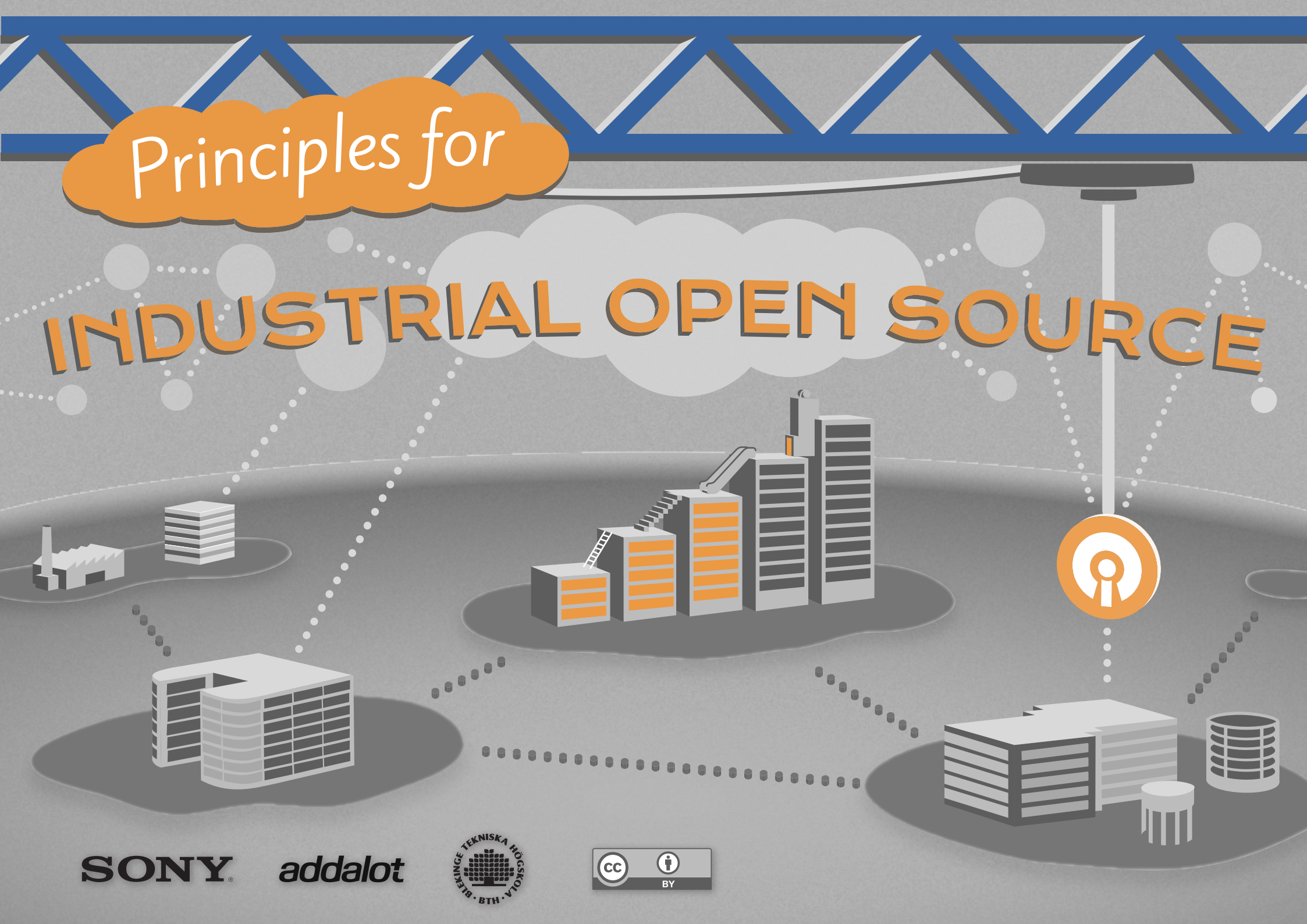
Principles for

INDUSTRIAL OPEN SOURCE



SONY addalot BLEKINGE TEKNISKA HÖGSKOLA · BTH ·

Principles for Industrial Open Source

Carl-Eric Mols
Head of Open Source
Sony Mobile

Nicolás Martín-Vivaldi
SW Management Consultant
Addalot

Magnus Ahlgren
SW Management Consultant
Addalot

Morten Werther
SW Management Consultant
Addalot

Krzysztof Wnuk
Associate Professor
Blekinge Institute of Technology

Illustrations and design: **Ove Jansson, www.monpetitstudio.fr**

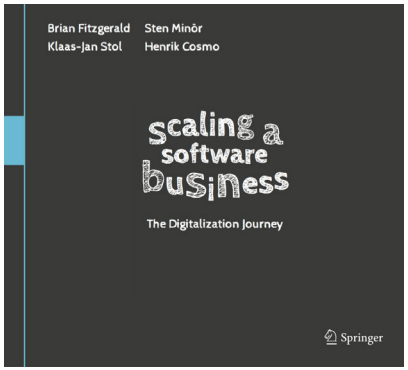
2018, Version 1.0, review version.
Please send feedback to feedback to **info@addalot.se** to enable a 2.0 version.

Lund, May 2018

Preface - Where it all comes from

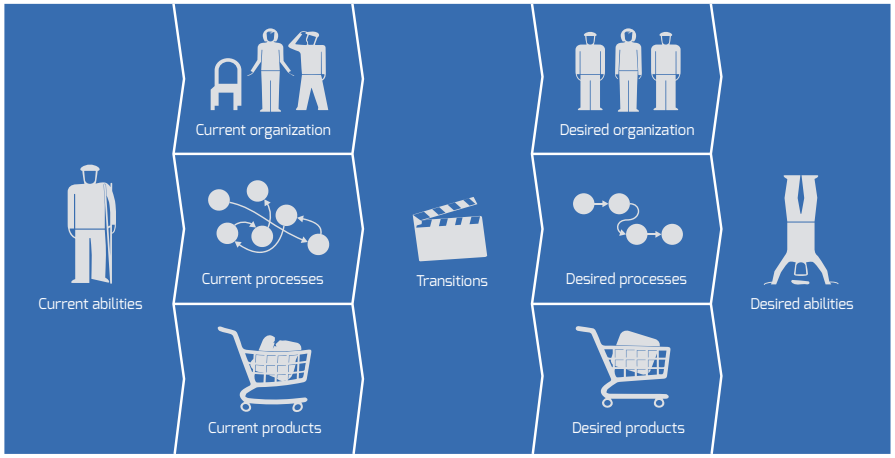
How do you transform your organization when software is becoming a critical part of your business?

This question was the main driver of the European ITEA2 project Scalare, a joint effort of industry and academia to tackle one of the key challenges in the European industry, the digitalization of industry and society.



The main outcome of the project is the Scaling Management Framework (SMF), which was presented in the book Scaling a Software Business in a format tailored for management. SMF is a model based on collected experiences from companies that have already made the journey to give software a central role within their organization.

The model is unique in that it suggests a holistic method to analyze and plan your journey. It claims that you can't focus solely on your products or services. You also have to look closely at your processes, your organization and how you define and decide on products. Inevitably, what goes on in these three dimensions will change as you increase focus on software. Central to the SMF is the canvas, a tool for organizing the analysis phase of the scaling project. In the SMF, general and reusable solutions are called patterns and are used



to guide you in reaching a desired outcome. The patterns that bridge the gap between the current and the wanted characteristics are the transformations the company needs to make.

The SMF and Open Source

The Scalare project produced two Open Source canvases covering standard patterns of successfully implemented transformations. The first canvas sprung out of basic, engineering driven reasons to use and work with Open Source. The second canvas was derived from more advanced, business driven reasons and it required the first canvas to be more or less fully implemented. Although the two canvases captured the high-level transformation needed to go from an almost Open Source ignorant organization to a full-fledged player in the industry, it still lacked both breadth and depth – and so the idea for this booklet was spawned. Parts of the Scaling Management Framework can be seen in that the three dimensions Organization, Process and Product are mentioned and that transformation solutions are called patterns, but the model for an Open Source transformation that is presented here is completely revised and can safely be read without any prior knowledge of the SMF.

1. https://itea3.org/project/scalare.html
2. The book *Scaling a Software Business* is open access under a CC BY 4.0 license. Download it at http://www.springer.com/gp/book/9783319531151

Most innovative software is Open Source

Facing a complex business environment where systems are no longer exclusively internally built, the industry has started to mix in-house, third party and Open Source software. Why, one might ask?

> **"IT leaders must understand, embrace, manage, and direct how and where Open Source will play a role in their strategic IT roadmaps to maximize the business value and minimize the risks associated with these technologies."**
> **– Gartner**

Open Source is the key to increasing development speed and lowering cost while boosting innovation. It's as easy as that. So, the next obvious question is: how? Well, this is exactly where Industrial Open Source comes in.

> **"If Software Is Eating the World then Open Source Will Chew It Up (And Swallow)"**
> **– Adrian Bridgewater, Forbes**

The statement above was written in 2015 and it has since then been proven over and over again. New technology adoption and penetration is considerably accelerated thanks to Open Source.

Open Source software can provide significant benefits to an organization. Many of today's fast-growing companies like Amazon, Google and Netflix as well as traditional industrials like Bosch, Porsche and Philips, have embraced Open Source as part of their business strategy.

In contrast to locking in their IP assets, they are making vital software assets public and free for anyone to see and use. How do they (dare to) do it?

Good news: it doesn't have to be an either-or decision. It's possible to simultaneously support Open Source while keeping parts of the code proprietary.

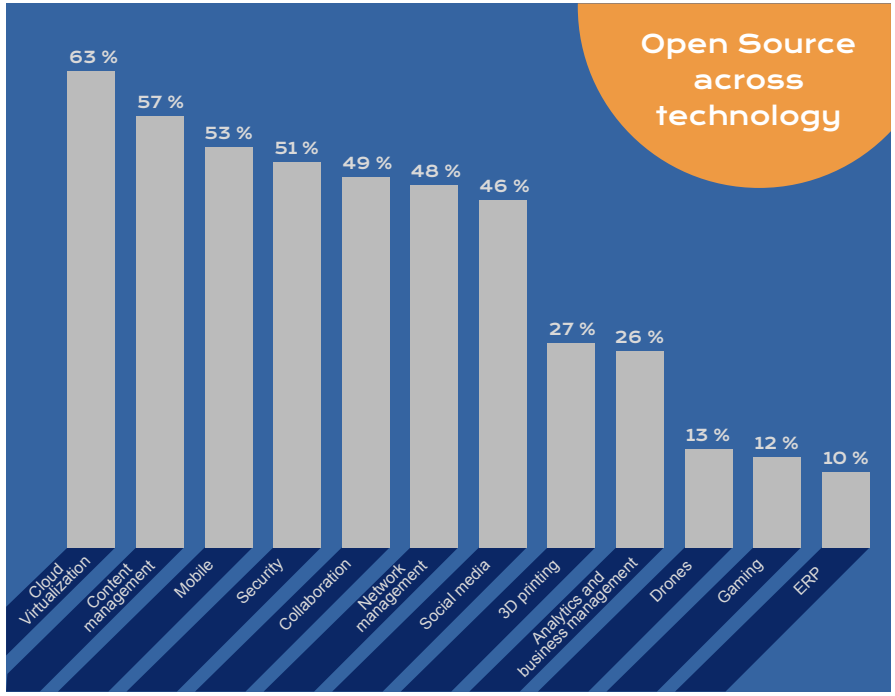
This booklet presents Industrial Open Source, industry proven and standardized patterns for how to manage an Open Source transformation. Use it as a guide for an Open Source journey. In particular you will learn:

- How to create more business – through new and alternative revenue streams.
- Why contribution is vital – to secure that value is added to your products
- Why compliance is a necessity – as your ticket to participate.

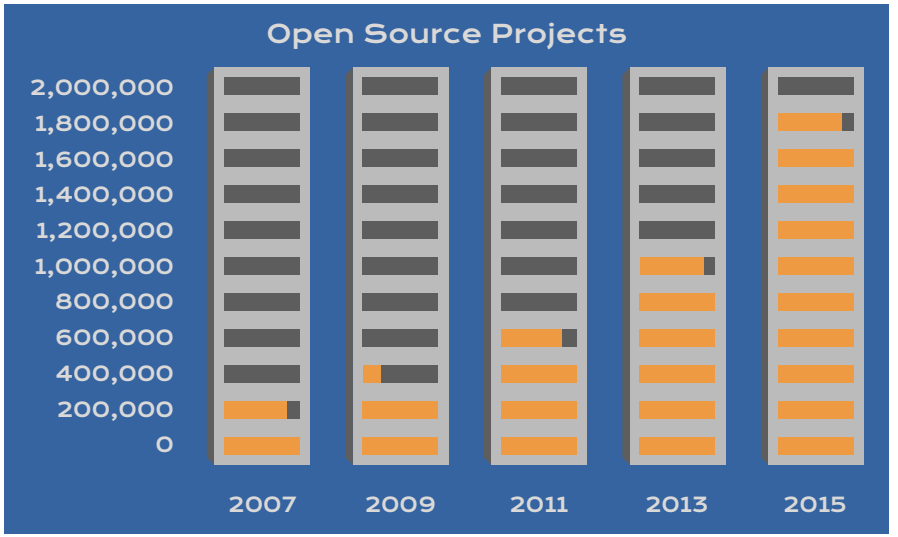
Have a safe journey!

Open Source is everywhere

The Open Source movement is several decades old, but it wasn't until the turn of the millennium that major companies entered the game. Apple is basing all their devices on Open Source since the release of OS X in 2001. The Linux OS, one of the most widely collaborative projects in the history of the world, is powering a substantial part of the computing power in the world - like web servers, mobile devices and supercomputers. Major web companies such as Google and Amazon are extensively leveraging Open Source dynamics. Data from a 2016 Black Duck audit of 1000 commercial applications shows that Open Source components were found in 96% of the applications.



Today, a majority of the fastest-growing companies base their offerings on Open Source. It is believed that in the long run, it is going to be impossible to succeed as a technology vendor without deeply embracing Open Source.



The huge toolbox of accessible technologies released as Open Source has become an enabler in a diversity of industries. Open Source is present in all domains, providing an enormous ecosystem of innovators that are sharing knowledge and creating new resources and opportunities for everybody to benefit from. This is the beauty of Open Source.

There are many reasons for why Open Source software has become so popular, like:

- Awareness of and access to global development.
- Open Source standards and practices have matured considerably.
- An increased demand for reduced development cost and shorter lead-times.
- Merge of domains, for instance connectivity technology with previously closed applications.
- The snowball effect: Since the big players are doing it, everyone else wants to get involved.

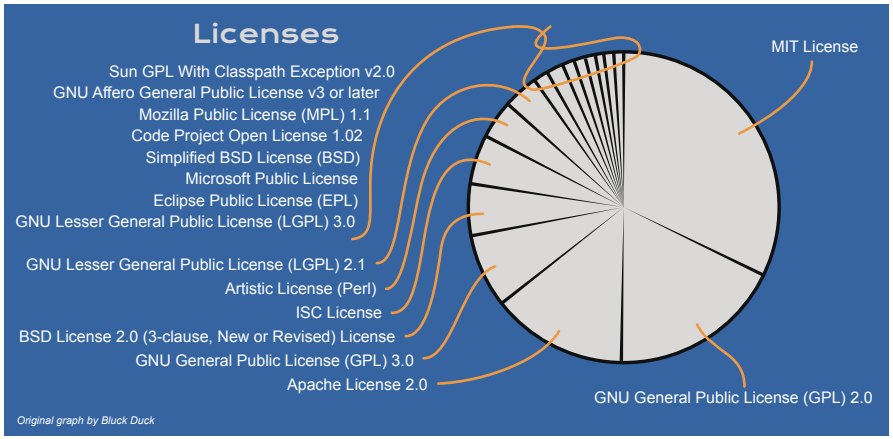
It could thus be argued that Open Source software is a central part of the digital transformation that industries around the world currently undergo.

Open Source and copyright

A subject that is traditionally a concern for many companies is the legal matter of software rights. This is especially true when it comes to how to safely combine Open Source software and proprietary code. Generally, Open Source software comes with permission to use, copy and distribute, either as is or with modifications, and may be offered either free or with a charge. Any modifications may have to be made publicly available.

The original creator owns the copyright and provides the precise terms in the license that follows the software. Without the license, the software automatically falls under general copyright laws, prohibiting us from copying, adapting and distributing it. With the license, we are granted certain rights to use the software.

The Open Source Initiative has to date approved over 80 licenses (http://www.opensource.org/licenses/). These are either permissive or so-called copyleft licenses.



The permissive licenses, also known as academic licenses, include BSD (Berkeley Software Distribution), MIT (Massachusetts Institute of Technology) and Apache. These grant the majority of rights to the licensee and allow for unlimited redistribution for any purpose as long as its copyright notice and the licensee's disclaimers of warranty are maintained. Derivative work, adaptations we make, can be distributed freely without restrictions.

All copyleft licenses derive from the original GNU General Public License (GPL) under which the free operating system GNU (including the Linux kernel) is licensed. The variants of this license are plentiful and span from giving away all rights to just a few rights. The general idea for all variants is to make software freely available.

This freedom comes at a fair price. All derivative source code has to be made publicly available and must be provided under their original licenses. It is, in other words, important to understand what derivative work means and how to combine proprietary source code with Open Source code.

Open Source potential

The potential and capability of Open Source software is unquestionable. Open Source runs operating systems, apps, databases, cloud computing, big data, and much more.

The benefits are clear: the industry-standard cost per line of code (LoC) ranges from $10 to $20, and the average component used by a Global 2000 company contains 50,000 lines of code. Therefore, the use of Open Source could in theory save from $500,000 to $1 million per project.

The cost advantage of Open Source may initially have been the main driver of its adoption, but is not the key benefit. It has proven to outperform proprietary software also on quality, security, reliability, and customization. The full benefits of Open Source are only realized when an organization has active control over its use. There can indeed be some serious problems with Open Source, but most often the potential gains outweigh the risks, which is why Open Source software has earned such a dominating position in the software landscape of today.

Benefits

Cost – Sharing the development and maintenance effort as is done in an Open Source community, substantially reduces the overall operating expenses.

Speed – Open Source solutions that are available off-the-shelf and are evolving at high speed can considerably reduce the time for an offering to reach the market.

Innovation – Companies that use Open Source get access to novel and innovative software, as well as enjoying participation in communities were this innovation occurs.

Business competition – As mentioned: cost, speed and innovation. But in addition, active involvement in Open Source communities can drive business and competitiveness through servitization and ecosystems.

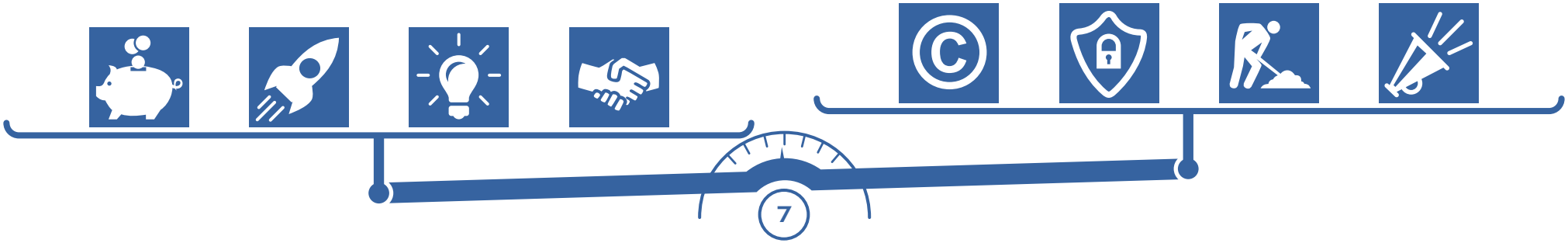
Challenges

Legal – Ignoring license compliance can result in copyright infringement, stop shipment orders, and immediately impact revenue streams.

Security – As with any software, managing application security is essential. Thus, Open Source software needs to be regularly monitored and updated to decrease the risk of security vulnerabilities.

Operational – Proper compliance is fundamental and requires the organization to train personnel and set up governance. On the next level, the operational barrier is about how to become a more active participant in Open Source communities.

Bad publicity –Not meeting Open Source compliance requirements may additionally result in bad publicity and damage the company reputation.

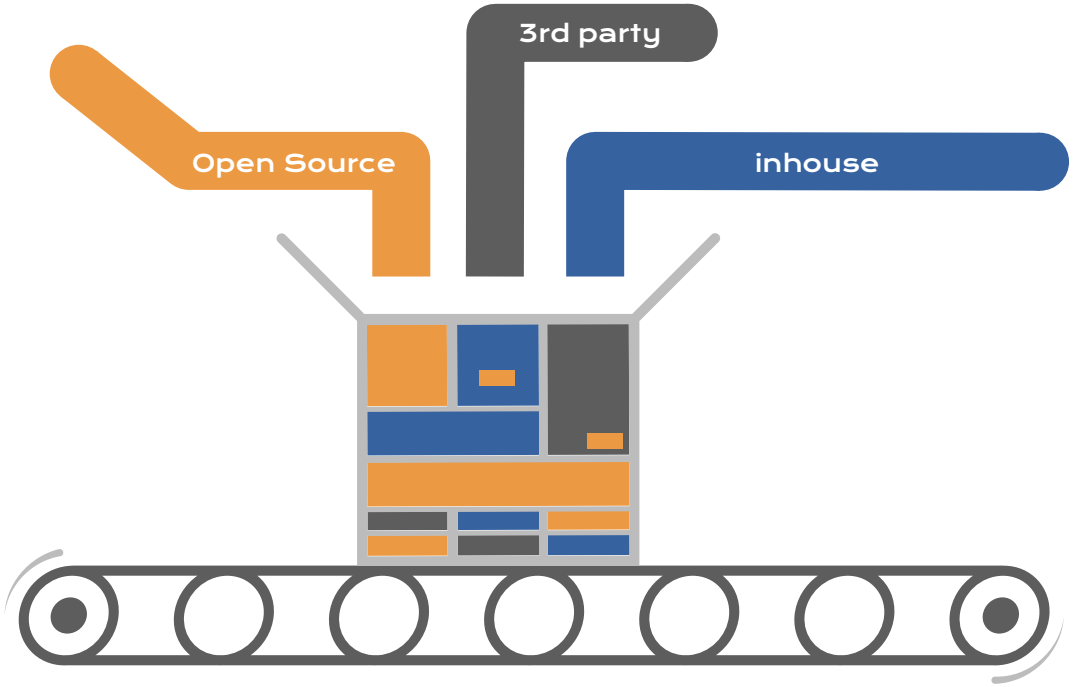


Industrial Open Source

This booklet targets a new wave of Open Source development being led by industrials and companies that are growing with Open Source at the heart of their business. These companies are using Open Source to build commercial products. They are creating new business models allowing them to succeed in emerging business domains using technologies such as AI, Cloud and IoT. Consequently, today's industry faces a complex environment where systems are no longer primarily built on proprietary development, but on a mix of in-house, third party and Open Source. Software companies are moving from development to integration.

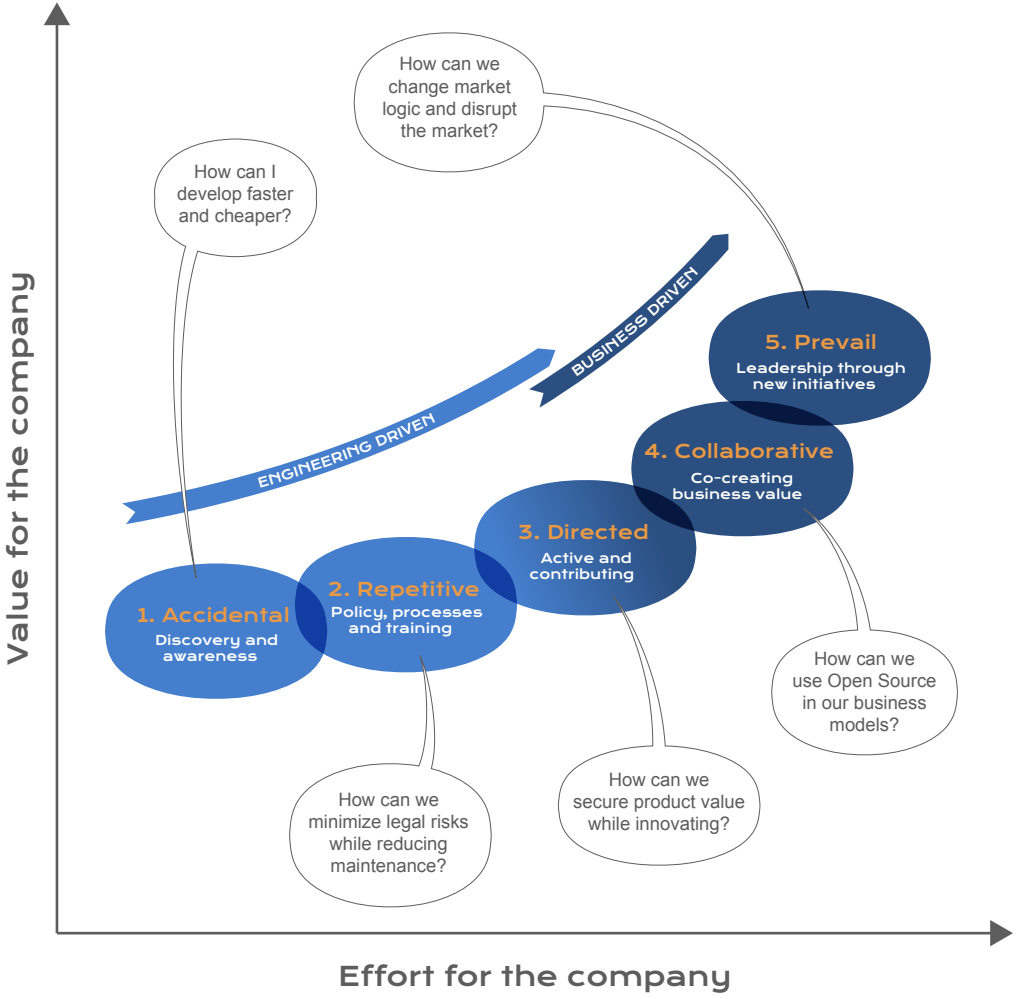
In the Open Source 360° Survey conducted by Black Duck in 2017, 60% of respondents said their organizations' use of Open Source increased in the last year, but an equally large number also indicated that they don't have a formal (policy and governance) process or are unaware of one in their organizations.

To manage risks and challenges and to help avoid unnecessary experimentation, organizations must apply an industrial approach to the use of Open Source as part of their software strategy, enforcing standardized patterns.



A journey with several steps

Successful introduction of Industrial Open Source is not a quick fix, it is a journey with several steps. This journey is divided into five levels of Open Source maturity, as shown in the diagram. A maturity level characterizes a company's Open Source capabilities. The levels are not distinct but overlapping, and provide a generalized understanding of the company's behavior.



Level 1 – Accidental

At the first level Open Source is an "under the radar" activity. Management position is often unclear or there might even be a policy against Open Source, but developers still use it based on own belief that shortcuts can be taken so that development will be faster and cheaper.

Level 2 – Repetitive

When management has realized the potential of Open Source but also the risks with not being Open Source compliant they push for a controlled repetitive framework - including policy and procedures. At this level Open Source is primarily used in an intake format focusing on cost and speed with limited contributions, mainly bug fixes that are made to reduce maintenance.

Level 3 – Directed

The value of collaborating with partners and competitors in the development process is understood and the company begins to champion specific projects and an Open Source approach in general. Contributions are done in line with an Open Source based product strategy and own industry experts are fostered. At this level, focus is on efficient and innovative product development but Open Source is still mainly an interest of the engineering department.

Level 4 – Collaborative

The understanding and use of Open Source now spreads from the engineering domain to encompass other domains like sales and business development. Alternative business models are used to capitalize on Open Source, and through ecosystems new revenue streams are exploited. Services complement the core product. At this level the focus is on business opportunities that can be harvested with the use of Open Source.

Level 5 – Prevail

The company has developed a full-fledged Open Source company culture, with full strategic support from the management to the extent that the company is able to disrupt entire markets by changing the market logic.

Level 1
Accidental

At the first level Open Source is handled under the radar by the developers. Management has not understood or are even in opposition to Open Source but developers still use it based on own belief that shortcuts can be taken so that development will be faster and cheaper.

Even if the work with Open Source preferably ought to be sanctioned by management it often starts as individual decision by developers who are aware of Open Source software components and use them for efficiency reasons, not to re-invent the wheel.

Another starting point is that Open Source has been included through integration of third party solutions.

Open Source software is particularly appealing for development organizations with a large heap of legacy and proprietary code that hasn't been maintained and refactored for a long time.

So, the engineers decide to use Open Source as a way to take shortcuts when solving development challenges, they have at hand. This is very common, but of course not the preferred way forward. Quite likely, they don't have formal approval from management.

The reason they don't include management and keep doing skunk work may vary. Management might use the not-invented-here argument, but more often there is a distrust of Open Source code. Until only a few years ago, Open Source was considered by most as a hacker's phenomenon and a headache for the Legal department. Neither are they likely to understand how to truly leverage from Open Source as long as they only consider it as being "free as in gratis" code. Both development and management need to learn more.

Level 2
Repetetive

At the second level Open Source becomes a topic on the development management agenda. To establish control, Open Source governance! Main drivers to reach level 2 are to increase the speed and to reduce the cost of development and at the same time fulfill the license obligations.

At level two management has understood the benefits (i.e. reduced cost, increased speed and increased innovation) of Open Source, and that these benefits are greater that the compliance risks (operational, legal and security). Together with the legal department, management has derived a directive or policy defining a common direction for Open Source. It describes the why, how and when Open Source software should be used.

Open Source governance processes and tools are established and followed. Three fundamental processes are essential; an Intake process for approving Open Source code that development intends to use, a Compliance process for ensuring that code follows the terms and conditions of Open Source licenses, and a Contribution process for the approval of code to be released to an Open Source community. At level two the focus is on using existing Open Source, contributions are mainly bug-fixes. A code scanning tools is almost a necessity to ensure governance control.

Successful Open Source governance require cooperation across many functions within a company, like legal, engineering and management. To ensure governance, organizational support and to drive improvements of the Open Source capabilities (Open Source maturity), an Open Source Board forum is established. There, all the involved disciplines and functions can meet. A coordinating role, often called Open Source Officer, is introduced.

At level 2 the company has also established Open Source code management, which means awareness of what Open Source is included in the code base and in which components. Efficient Open Source usage and code management require a modularized code base to facilitate integration of open Source components and maintain compliance.

A challenge to fully using Open Source is the reluctance to losing control as a result of including Open Source components into the product. To get to level 2, start with motivation and control. Management needs to show commitment by formulating a clear direction. Control starts with defining the current baseline of existing Open Source in the product supported by a process for compliance including a scanning tool. Another key is to introduce the new organizational forum and roles – to drive the Open Source journey.

Open Source is a key ingredient to product development at the third maturity level. Focus is on creating product value, by having a clear view of how to make the most out of Open Source in development

At level three the company has realized the value of contributions (reduction of maintenance cost, reduced time to market, and increased influence on communities)

The goal is not to contribute everything, but to base it on a Make-Buy-Share strategy. This way the company can focus on the differentiating components while using sourcing or Open Source software for the other parts of the system. This strategy, will increase innovation across the product through better-focused own development and joint development of qualifier and commodity components. All Open Source software require a Contribution strategy that will ensure that developers don't waste time and effort on contributions that are not in alignment with the strategic ambitions and thus are not likely to be approved.

Since parts of the product now are based on Open Source, the company will no longer have full control of the roadmap for its product. Through more active engagement in communities, the company can regain control to some extent by influencing development. Still, the product strategy of the company needs to be transformed into a collaborative model better adapted to the distributed nature of working with Open Source.

In order to influence communities, the company needs to put effort in fostering industry experts. This is done by building on Open Source culture, highlighting contributions and providing both individual support as well as driving internal communities.

Many software-intensive companies own software assets that are of general interest and high potential but do not contribute to the product differentiation. These assets are suitable to create Open Source communities around, with the objective to harvest the advantages of community involvement e.g. increased innovation, decreased cost and time to market.

To get to level 3, focus on preaching the value of contributions and derive the Make-Buy-Share strategy. The key is to get involvement from Product Management, who also often has the belief that the proprietary codebase constitutes huge value and are concerned that they will lose control, which further restricts contributions.

At the fourth level, Open Source is not only an enabler of engineering goals, but is used to create business gains. This is achieved by collaboration in ecosystems based on Open Source, including partners, customers and end users.

Now, Open Source is no longer only an engineering discipline, but a game changer to the business. The company has realized that Open Source adds additional business models that enable:

- Accelerated growth of business – to expand the market in terms of a broadened offering and to grow higher in the value chain.
- Disruption of market entry barriers – to gain access to a market with an open offering, while raising the bar for proprietary and non-collaborative businesses.
- Opening for alternative business opportunities – to benefit from alternative revenue streams like ecosystems.

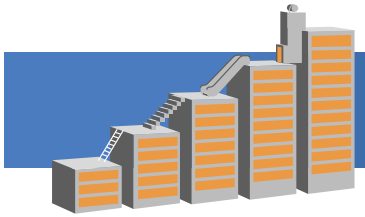
There are three alternative business models, the extended business model (when alternative revenue is collected from something related to the core offering, e.g. a service fee), the indirect business model (when revenue is mainly collected through a device or a hardware offering) and the asymmetric business model (when revenue is collected from a source unrelated from the core offering, like data or ads).

The company is able to identify opportunities, create and orchestrate an Open Source based ecosystem. An ecosystem can be described as a community of communities and require a collaborative business model with for example revenue sharing. Ecosystems also require an Open Sources based platform where the Ecosystem players can add services and contribute to the platform. The core product is now a part in a service offering.

To support this, the company is now based on self-managed teams supported by a visionary leadership and is much more suited for operating in a complex, collaborative and dynamic environment as the Open Source world. This type of company will fully benefit from an agility to identify and harvest new business opportunities and using Open Source communities and ecosystems to realize them.

At this stage the company is well versed on the engineering and legal aspects of Open Source. The company's knowledge level on Open Source is satisfying to the extent that directives and policies are relaxed and some automation of the governance processes has been introduced.

To reach level 4, Open Source needs to grow outside engineering. Management will have to explore radically different market logics. They must question what the core offering of the company is and consider how alternative revenue streams can be created.



Level 5
Prevail

At the fifth level the company has developed a full-fledged Open Source culture with complete strategic support from top management, to the extent that it is able to disrupt entire markets.

Companies at this level design products to be heavily based on Open Source, thus obtaining a competitive advantage by exploiting the full innovative strength of ecosystems. They are launching major technical innovations to be shared by Open Source communities, thus setting de-facto standards and becoming known as an authority.

Open Source practices are mastered to the extent that these companies are able to develop Open Source communities to complete ecosystems. Level 5 companies are seen as the prime provider of Open Source code, tools and project hosting in their industry domain, thus able to orchestrate the development of industry wide initiatives (an example being Google with Android).

Open Source has become a core competence and most of the staff are skilled in this area. A few even receive worldwide acclaim as prominent names within Open Source.

By relying on Open Source, the company has the power to disrupt and redefine the market logic and how value in the market is created and captured. Through Open Source initiatives it can control its industry and take the lead, as well as create entirely new value propositions.

Management and staff live and breathe Open Source. Open Source is entrenched through the company walls and communities outside the company recognize the company as being in the forefront and gurus of Open Source. Management does not only encourage Open Source projects and initiatives but requires employees to drive Open Source for the benefit of both the company and the ecosystem.

Patterns – the pieces of the puzzle

Our definition of a pattern is adapted from the "Software design pattern" definition, that is, a pattern is a general, reusable solution to a commonly occurring problem within a given context. It is not a complete solution that can be transformed directly into an organization. It is a description or template for how to solve a problem that can be used in many different situations.

Industrial Open Source contains 27 different patterns that are carefully selected and crafted to bridge the gap between the current and the wanted characteristics of the transformation the company needs to make. Patterns can be seen as the pieces of the puzzle in getting a complete picture of your Open Source transformation journey.

Patterns are sorted in three different types: organization, product and process.

The **organization type** includes aspects needed to successfully drive Open Source: Which roles and functions to establish. What culture and organization structures are needed to make it happen and how to include Open Source in company products and business strategies.

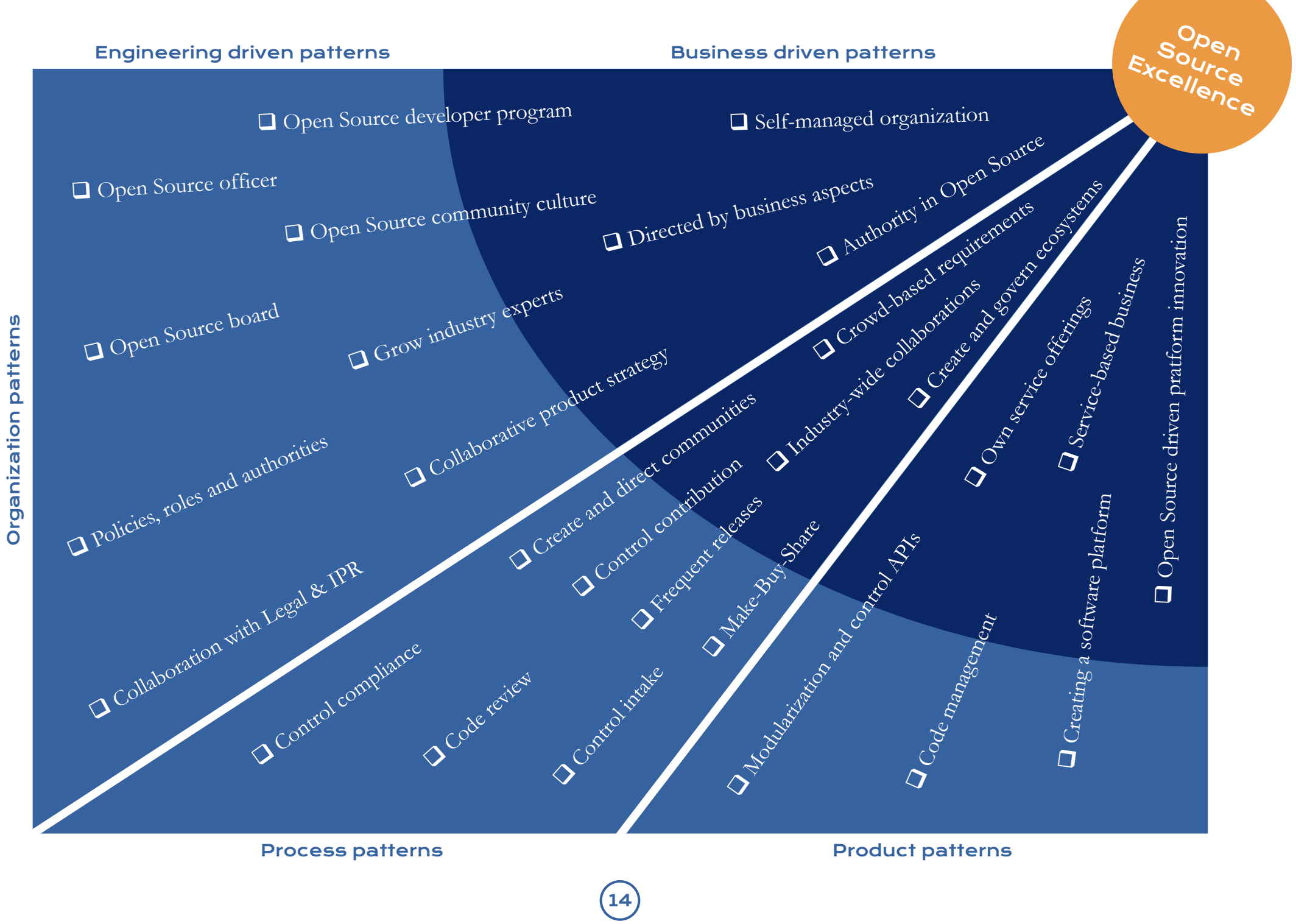
The **process type** covers the activities needed in Industrial Open Source. Activities include Open Source governance (like compliance, intake and contributions), product management (like make-buy-share) and how to create and direct communities and eco-systems.

The **product type** focuses on products and services, how to structure these (architecture), and how to make the code base available. Patterns are e.g. about understanding what Open Source components the company has in its system and how to establish modularized reusable platforms with added services.

Most patterns have a dependency to other patterns. Some patterns are basic pieces that need to be put in place early like "Control Compliance". Others are more advanced, like "Open Source Driven Platform Innovation" that build on the basic ones. Some patterns are of a general nature; practices that any development organization would benefit from, (like "Code Review" or "Frequent Releases"), while others are unique to Open Source. The order in which the patterns are implemented must however be based on the situation (strengths and weaknesses, etc.) in your organization. They are all essential on your transformation journey to successfully incorporate Open Source.

An overview of all patterns is presented on the next page. The patterns are organized according to the three types and whether they are primarily engineering driven ones (lighter blue shade) or primarily business oriented driven ones (darker blue shade).

To get a rough understanding of how the patterns relate to the different Open Source maturity levels, see the picture on page 20. However, it is important to point out that the order should not be seen as a recommendation for how to implement them.


Engineering driven patterns
Business driven patterns
Open Source Excellence
Organization patterns
Open Source developer program
Self-managed organization
Open Source officer
Open Source community culture
Directed by business aspects
Authority in Open Source
Open Source board
Grow industry experts
Crowd-based requirements
Policies, roles and authorities
Collaborative product strategy
Industry-wide collaborations
Create and govern ecosystems
Own service offerings
Service-based business
Open Source driven pratform innovation
Collaboration with Legal & IPR
Create and direct communities
Control contribution
Frequent releases
Make-Buy-Share
Control compliance
Code review
Control intake
Modularization and control APIs
Code management
Creating a software platform
Process patterns
Product patterns

Taking the steps

All improvement initiatives require Change Management. Change Management is the discipline that guides how we prepare, equip and support individuals to successfully adopt change to drive organizational success. An Open Source journey will encounter many challenges that require change management.

Introducing Industrial Open Source with all its patterns is a true change management challenge. To succeed, an Open Source Program with dedicated resources supported by executive management is required.

General recommendations for Change Management

Use business goals as drivers and ensure that improvement activities are connected to theses drivers. For example, an Open Source Program might have goals on reduced lead-time, reduced cost and improved innovation and these improvement objectives must have hooks on the business goal level.

Involve key practitioners when establishing the Open Source processes. Don't leave it only to pure process expertise or to external people not aware of existing best practices. A mix is often the best, internal expertise aware of current capabilities together with external experts providing industry and change management experience.

Drive improvements in a lean way, i.e. reduce the amount of parallel work. Focus on a few improvements at a time and get them in place, since spreading the resources and efforts too thin will risk not finishing anything. In an Open Source Program context, get the Open Source Board in place early, since it can serve as the engine for the improvement activities.

Don't aim for the advanced patterns before you have the basics in place. You must walk before you can run. For example, focus on getting a Make-Buy-Share strategy in place before investing in building Ecosystems. See the pattern overview for a rough definition of how patterns relate to the different Open Source maturity levels.

Remember the human side. Involve the people. Change starts with people and then continues throughout the organization. The pattern "Org-5 Open Source Community Culture" is important for developer acceptance and buy-in.

Understand thebenefits of Open Source code

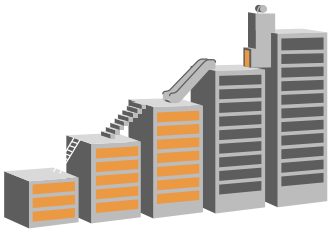
The first step on the way to engage in Open Source is often driven by developers with the belief that this makes development faster and cheaper. This usually happens without management awareness or consent. The first challenge is to realize that your software system is likely to contain Open Source software (put there by your developers and through integration of third party solutions) and now it should be driven out of the shadows – by appreciating and clarifying its benefits.

Starting on level 1

Acknowledge the compliance obligations

The picture of the benefits of Open Source needs to be complemented with an understanding of how to manage Open Source compliance. This leads the organization to introduce a governance, implying more structure and control. It's important to establish for all involved stakeholders that this is the "entry ticket" to proper Open Source usage. A challenge to fully using Open Source is the reluctance to losing control as a result of including Open Source components into the product. To enter level 2, management must not only accept Open Source, but also approve and commit to it.

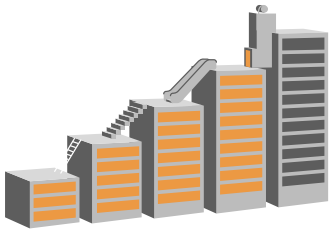
Going to level 2



Realize the value of contributions

Many organizations get stuck on level two. Using Open Source saves time and effort, but the contribution process is seen as complicated and costly and the value of it is not understood. Often there is also a belief that the proprietary codebase constitutes huge value and product owners are concerned that they will lose control, which further restricts contributions. Main change management challenge is to establish an understanding of the importance of contributions and participation in Open Source communities (Cost reduction, Time to market and Innovation). See the Anti-Pattern "Anti-1 Shun the "Use, but not contribute" trap" for more details.

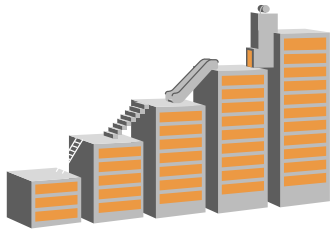
Going to level 3



Not only for Engineering

Open Source is often viewed as an engineering concept. Going to level 4 requires the whole organization, but primarily sales & marketing and business development, to understand how Open Source can be used to create novel business opportunities. By controlling and directing an Open Source community, you commoditize and standardize technology while keeping the advantage of timing, cost and innovation compared to competition. Through an ecosystem new revenue streams can be exploited. The main challenge is to create an organization where business decisions and engineering is done in close cooperation – preferably in the same team!

Going to level 4

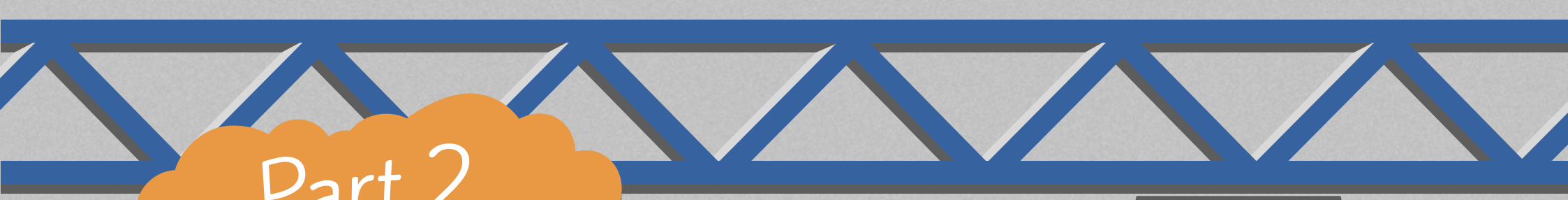


Practice makes perfect

By doing level 4 repeatedly your learnings increase, and your impact will be greater. The challenge is to maintain the belief that Open Source and Ecosystems is a business enabler and secure that the new business models are successful by constantly monitoring and changing with the market! At some point you might even have disrupted the marketplace. Then – you have reached the end of this transformational journey.

Going to level 5

Congratulations!



Part 2

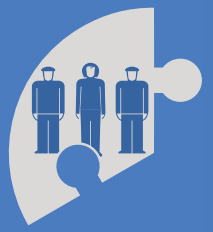
The Industrial Open Source Patterns

Industrial Open Source Patterns

After gaining a general understanding of the benefits of introducing Open Source in your company, it is the time to get hands-on with the different patterns.

All patterns contain a brief description including the problem they are solving along with implementation details and benefits, and a reference to further reading. Note that there is no recommendation to read the patterns in a specific order. It is better to get an idea of your current maturity and how the related patterns are fulfilled. Most companies, no matter what previous experience they have from Open Source, are likely to lack some parts of the basic level 2 patterns. It is strongly advised to increase Open Source maturity in steps – see "Taking the steps".

All patterns are more or less connected and applicable throughout the Open Source journey. A rough indication of how patterns relate to the different Open Source levels is provided on the next page. It gives a general suggestion of which patterns to implement to get to a certain level on the maturity scale.



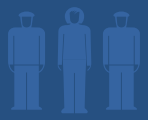
Organizational patterns, which are dealing with organizational matters such as structures, roles, competences and strategies

Process patterns, which are focusing on activities to keep in control, like intake, compliance, and contribution

Product patterns, which are involving aspects related to the software and architecture of the products and services

Anti-patterns are common pitfalls that companies may get stuck in at a certain point

5. Prevailed
Leadership through new initiatives



4. Collaborative
Co-creating business value

ORG-9 Self-managed organization

ORG-11 Authority in Open Source

ORG-10 Directed by business apects

PRC-8 Industry-wide collaborations

PRC-10 Create and govern ecosystems

PRC-9 Crowd-based requirements

PRD-5 OS driven platform innovation

PRD-6 Service-based business

3. Directed
Active and contributing

ORG-5 Open Source Community culture

ORG-7 Open Source Developer program

ORG-6 Grow industry experts

ORG-8 Collaborative product strategy

PRC-4 Control contribution

PRC-6 Frequent releases

PRC-5 Make-Buy-Share

PRC-7 Create and direct communities

PRD-3 Creating a software platform

PRD-4 Own service offerings

2. Repetitive
Policy, processes and training

ORG-1 Cooperation with legal & IPR

ORG-3 Open Source officer

ORG-2 Policies, roles and authorities

ORG-4 Open Source board

PRC-1 Control compliance

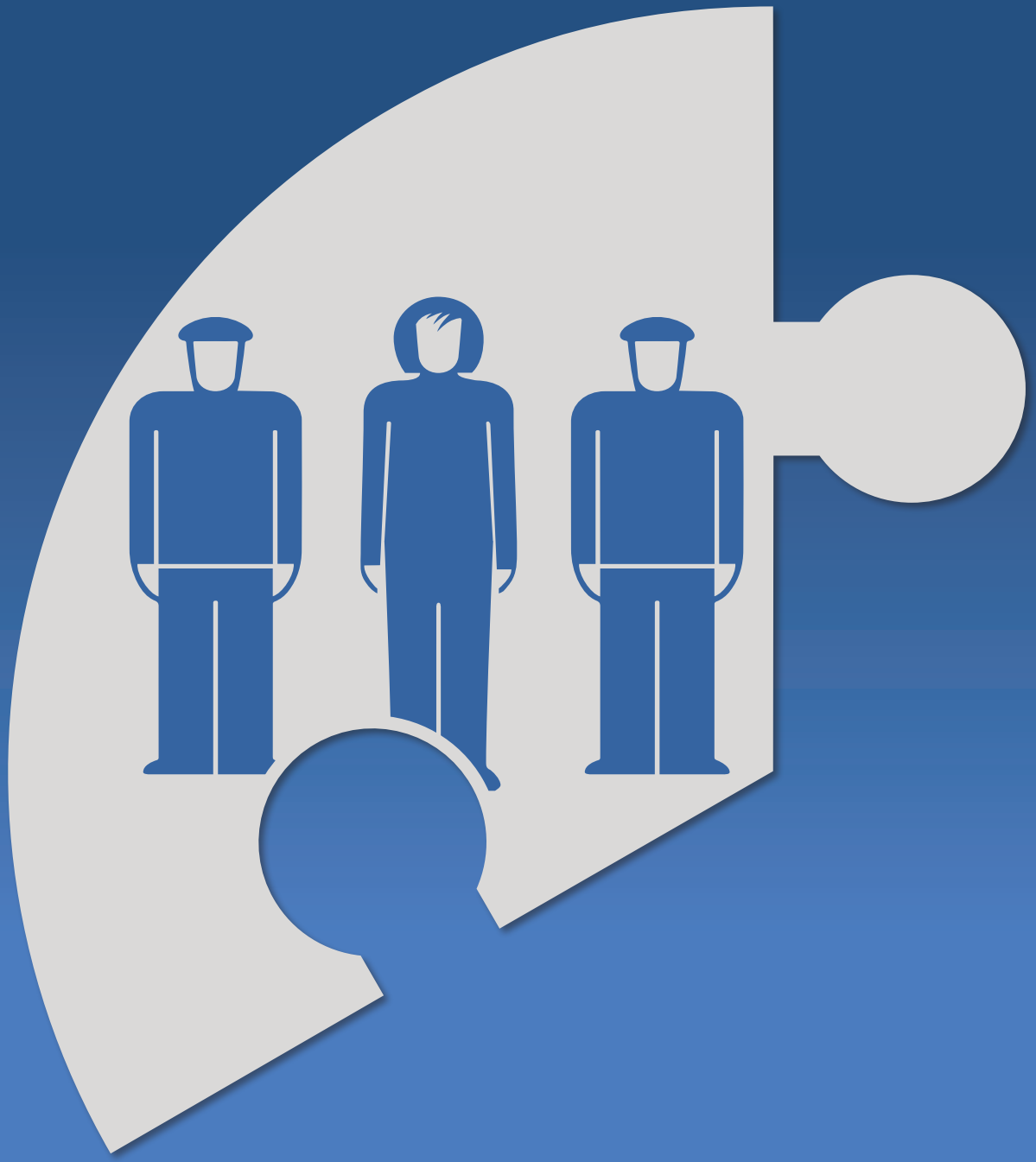
PRC-3 Code review

PRC-2 Control intake

PRD-1 Modularization and control APIs

PRD-2 Code management

1. Accidental
Discovery and awareness



Organization Patterns

Collaboration with Legal and IPR

> Several problems can arise if you do not follow legal terms and conditions, but equally there may be lost opportunities from being overly cautious.

To manage both the day-to-day operations of Open Source and the longer term strategic issues, all the involved stakeholders need to get a better understanding of legal aspects (like IPR matters, Open Source licenses and compliance). As a starting point, this means finding a way to cooperate between management, software and Legal and IPR.

What it covers

Legal aspects are an important part of understanding how to properly implement Open Source in a company. Several problems can arise if you do not follow legal terms and conditions (e.g. see pattern Proc-1 Control Compliance), but equally there may be lost opportunities from being overly cautious – maybe to the extent of refraining from using Open Source code.

Management of Open Source covers a broad palette of tasks that to some extent have a need for a legal advice:

- Processes: How to manage intake, compliance and contributions so that community licenses and company IP is taken into account.
- Policies: Directing the company around Open Source and the different processes connected with it, based not least on a sound understanding of legal facets.
- Open Source strategies: Covering the long term objectives for Open Source and what is required in terms of organizational and business development to reach them which often lead to questions on IPR.
- Knowledge base: When questions arise around legal aspects of Open Source, there needs to be someone or somewhere to turn to for support.

To accomplish the above whilst respecting legal aspects, the main stakeholders in the company – management and software – need to involve Legal and IPR. In its simplest form this can be done by setting up regular meetings and assigning legal roles, like a counselor, to support the development company with easy access to legal advice. A key ingredient in this collaboration is the Open Source Board (see pattern Org-4) and the Open Source Officer (see pattern Org-3).

Why it is important

To get proper traction of Open Source adaptation within a company, there is a need to swiftly set policies and processes in place, and to correctly weigh in the legal risks that are involved in dealing with them (for instance compliance and contribution). This requires cooperation between stakeholders that do not normally have an arena for exchange – mainly management and software versus Legal and IPR. To accomplish successful Open Source operations, this cooperation needs to be started early on and thereafter needs to be maintained over time (for instance when considering when and how to start communities).

Considerations

As indicated above, it is important to start early by setting the rules for how the company should be working with Open Source, which means that it is prioritized to set up:

- Policies for Open Source.
- Processes:
  - Starting with intake and compliance where licenses are important.
  - Later, contribution where IPR questions are in focus.

To get policies and processes in place, the collaboration with Legal and IPR is absolutely essential. In order to make the cooperation efficient, ensure that exepectations, roles and responsibilities are defined from the very start (see Pattern Org-2 Policies, Roles and Authorities). Initially cooperation needs to be tight, but as the organization matures and things are settled, less frequent communication will be needed.

For a small company it may not be possible to hire full-time legal support, but then it should be considered to at least part time acquire this service from an external law firm. Not only if legal counsel is a scarce resource in the company, but especially then, there should also be legal training given to individuals at the center of Open Source management.

Rather than just setting up a semi-formal cooperation between development and Legal and IPR, it is recommended to move as early as possible to establishing the Open Source Board and the Open Source Officer role. This requires a clear definition of cross-functional cooperation points that are mentioned in this pattern.

Further reading

- I. Haddad “Free and Open Source Software Compliance The Basics You Must Know” http://www.ibrahimatlinux.com/uploads/6/3/9/7/6397792/0.pdf

Related patterns

- Org-2 Policies, Roleas and Authorities
- Org-3 Open Source Officer
- Org-4 Open Source Board
- Proc-1 Control Compliance
- Proc-4 Control Contribution
- Org-1 Cooperation with Legal & IPR

Policies, Roles and Authorities

> Without having a leading star it will hardly be possible to gain momentum in the Open Source journey. This is directed through an Open Source policy.

In order to succeed with Open Source, a company must provide a common direction including the high level answers to why, who and how. This should be packaged in an Open Source Policy that will govern the execution of the Open Source activities.

What it covers

The Policies, Roles and Authorities pattern is about having the necessary organizational fundaments in place to govern execution of the Open Source practices.

Policies are about having an organizational directive on the role of Open Source in the company and how Open Source will be governed. They should clearly state:

- Why Open Source is part of the company's software strategy.
- What is the role of Open Source and what is the intended direction.
- The key processes that outline how the company should work with Open Source:
  - Intake, how Open Source software should be taken in.
  - Compliance, how license compliance should be ensured.
  - Contribution, how contributions should be controlled.
- The necessary roles to execute Open Source-related activities and how Open Source activities will be governed.

There is a set of generic roles that are needed in one way or another. Since the role describes the position and expected behavior, one person may have more than one role and there may also be many people having the same role in the company:

- **Open Source Officer:** The main objectives of an Open Source Officer are to be the center for Open Source activities and to drive the company's ability in Open Source-based development and business, i.e. its Open Source maturity. This role is described in a separate pattern: Org-3 Open Source Officer.
- **Business Manager.** The Business Manager is a generic role representing any line manager having the authority to decide on matters regarding Open Source on behalf of the company. The company's structure and authorization rules will govern the exact mandate of business managers at different levels.
- **Intake officer:** The Intake Officer role is assigned by the Business Manager to draft or approve intake evaluations. The intake process is further described in a separate pattern: Proc-2 Control Intake.
- **Compliance officer:** The responsibility of the compliance officer (also appointed by the Business Manager) is to ensure that the company adheres to Open Source licenses throughout the software life cycle. This compliance process is further described in a separate pattern: Proc-1 Control Compliance.

- **Contribution officer:** assigned by the Business Manager, who is main responsible for the drafting of Contribution Proposal. The Contribution process is further described in a separate pattern: Proc-4 Control Contribution.
- **Legal subject matter expert:** Legal competence is necessary to act as subject matter expert regarding Open Source licenses' terms and conditions. This is further detailed in a separate pattern: Org-4 Open Source Board.
- **Intellectual Property Rights (IPR)** subject matter expert: IPR expertise in needed to provide guidance in matter concerning property rights such as copyright, patents and trademark. This is further detailed in a separate pattern: Org-4 Open Source Board.
- **Open Source Board:** The role of the Open Source Board is to maintain the open source policies, govern the execution of the Open Source practices and provide guidance and decision on matters concerning Open Source. The organization must ensure that the necessary roles and people are represented including management, Legal & IPR and development. This Board is described in a separate pattern: Org-4 Open Source Board.

Why it is important

Without having a leading star it will hardly be possible to gain momentum in the Open Source journey. This is directed through an Open Source policy that sets a collective direction ensuring that necessary activities are executed, and that the appropriate governance is in place.

Creating a common consciousness about why, how and who, gets even more important with increased maturity, since an Open Source oriented company aims for decentralized structures (see pattern Org-9 Self-managed Organization). Without a clear directive there is a risk of teams driving in different directions and business benefits get lost.

Considerations

Since this pattern is about implementing the organizational fundament for the Open Source strategy, it requires management commitment and buy-in. By being serious and making the necessary investments, mistakes will be avoided and successes will be granted.

The most important thing is to get the right people involved and engaged. In short, sufficient representation from development, Legal & IPR and management needs to be ensured. Since Open Source is an area with specific characteristics, the people need to be interested and dedicated to engaging themselves. If they believe in the cause, they will have the necessary self-drive. These people are well suited to write the policy and to form the fundament in the Open Source Board.

Communication and adoption are key. To get a Policy known and living it is instrumental to spread the word and carefully manage the process of implementing what is written in the policy.

Furthermore the policy needs to be updated throughout the Open Source journey. Initially at the lower levels of maturity, the focus will be more on control. Moving up the maturity ladder, focus will change to more business oriented practices. The policy will need to be to continuously adjusted to reflect these changes.

Bear in mind to keep the investment aligned with the scope of the Open Source strategy. If Open Source is only included in a limited part of the software, the Open Source organization should be sized accordingly. However, if Open Source is an integral part of the software strategy, the necessary involvement from affected parties at all levels must be ensured.

Further reading

- B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg. "Who is an open source software developer?." Communications of the ACM, February 2002, pp. 67-72.
- https://www.linuxfoundation.org/resources/open-source-guides/starting-open-source-project/

Related patterns

- Org-3 Open Source Officer
- Proc-2 Control Intake
- Proc-1 Control Compliance
- Proc-4 Control Contribution
- Org-1 Cooperation with Legal & IPR
- Org-9 Self-managed Organization

Open Source Officer

> The Open Source Officer needs to have a cross-functional perspective, acting at the center of activities related to governance and organizational support and development.

Proper Open Source compliance and operations require cooperation across many disciplines within a company, like legal, engineering and management. To ensure governance, organizational support and to drive improvements of the Open Source capabilities, the coordinating role of an Open Source Officer is needed.

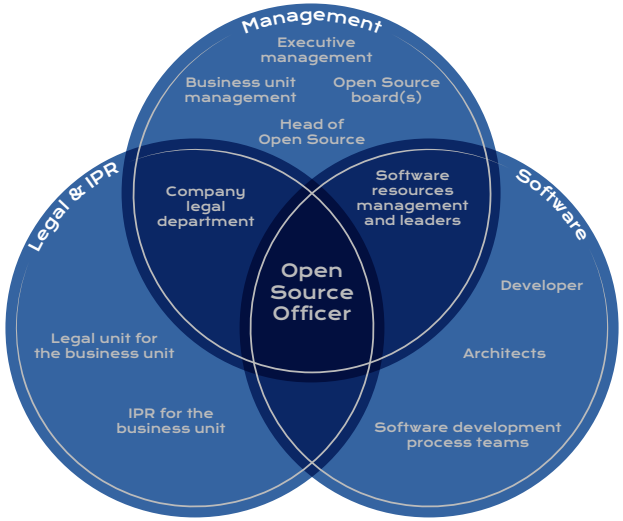
What it covers

Due to the multi-disciplinary nature of Open Source operations within a company, the Open Source Officer needs to have a cross-functional perspective, acting at the center of activities related to governance and organizational support and development – to secure alignment, progress, timeliness and compliance. This is done through responsibility for, or involvement in, the following areas:

Coordination between disciplines: Acting as an interpreter and discussion facilitator in matters related to Open Source between for instance legal and development engineers, between development management and sourcing or more generally between all involved domains and functions.

Training: Ensuring that relevant training is available for the staff. This may include participation in creating training material. Training needs must regularly be analyzed and assessed.

Open Source artifacts: Making sure that directives, guidelines, templates and documentation are provided and shared. This also covers processes and the need to evolve them over time.



Open Source Board: The Open Source Officer is an active participant of the Open Source Board (described in pattern Org-4) and partakes in the decision making there.

Open Source spokesperson: An Open Source Officer can act as an external interface for the company for Open Source related inquiries or function as an internal management consultant when Open Source is on the agenda in e.g. management discussions.

Apart from the abovementioned areas, it is also important that the Open Source Officer is working for the long-term objectives of Open Source within the company through active change management, suggestions for future improvements and networking in the Open Source community.

Why it is important

Open Source constitutes a paradigm shift for software development and with it follows a very specific set of logics, rules and cultural behaviors. For a company to fully take advantage of all possibilities it is important to secure the necessary domain expertise in the field. Without the support of an Open Source Officer, everything from compliance processes to driving more advanced maturity patterns will be considerably more difficult to accomplish.

In addition, there is also the consideration of the multi-disciplinary impact of Open Source in a company. Both organizational, hierarchical and technical boundaries are crossed, and it is essential to establish a role that connects and interprets between everyone involved. Already from the start there will be challenges in closing the gap between management, legal and engineering. The Open Source Officer can help to secure that things are not falling between the cracks.

Considerations

A crucial starting point for Open Source in a company is the understanding and commitment from management. Due to the bridging function of the Open Source Officer role it is central that 1) management reporting is done at an appropriate level with sufficient visibility, 2) the role is given necessary bandwidth and mandate within the company and 3) management shares its visions, continuously follows progress and supports the Open Source Officer when needed.

The skill set needed from an Open Source Officer is in a sense both wide and deep:

- Open Source licenses and obligations, development and governance.
- Knowledge and experience of driving process and change management on a companywide level.
- Sufficient technical understanding to discuss with engineers.
- Strong communications and leadership profile.

In very large companies, there may be a need for more than one Open Source Officer. The general recommendation is one Open Source Officer per site and business unit, with a coverage of up to 1000-1500 employees. Similarly, for a smaller company the Open Source Officer role might be covered by a part time assignment.

Finally, to be effective the Open Source Officer needs an established and well supported Open Source Board (see pattern Org-4), since this is the main platform for governing the organizational efforts on Open Source.

Further reading

- C.E. Mols, K. Wnuk, J. Linåker , "The Open Source Officer Role – Experiences" OSS 2017: Open Source Systems: Towards Robust Practices, pp. 55-59.

Related patterns

- Org-1 Cooperation with Legal & IPR
- Org-2 Policies, Roles and Authorities
- Org-4 Open Source Board
- Proc-4 Control Contribution

Open Source Board

> The Open Source Board effectively governs the evolution of Open Source and may appoint the Open Source Officer as a driver, for instance for cross functional activities where organizational ownership is unclear.

There is a need for a forum where all the involved disciplines and functions regularly can get together to align on matters concerning Open Source, drive the implementation of Open Source practices and resolve issues when needed. This is the Open Source Board.

What it covers

The Open Source Board is a forum where all the major stakeholders involved in Open Source activities within a company meet and discuss on a regular basis. The Open Source Officer (as described in pattern Org-3) is one of the permanent members of the group that also covers delegates from Legal and IPR, management and engineering.

The main mission for the Board is to define and support the Open Source journey for the company by 1) establishing policies and processes, 2) setting and monitoring key KPIs and 3) deciding on and governing improvement activities that increase the overall maturity and capability level. Through actions in these areas, the Open Source Board effectively governs the evolution of Open Source and may appoint the Open Source Officer (or any of the other board members) as a driver, for instance for cross functional activities where organizational ownership is unclear.

Other duties for the Open Source Board include:

- To be involved in the contribution decision flow, for instance in complex cases where the joint competence of the Board is called for or when IP conflicts may arise (for large and complex contributions, see Pattern Proc-4 Control Contribution).
- To be an escalation point for issues where there are conflicting interests in the company.
- To act as a knowledge base for Open Source and secure support and information sharing when needed.
- To be a consultation body for all (larger) decisions that are to be taken with respect to Open Source in the company.
- To monitor and support the Open Source introduction activities and other needs in an company, e.g. training or competence requests reported by the Open Source officer.

Why it is important

Since there are many stakeholders for Open Source within a company, it is vital to create a forum where all can meet and agree on how to progress. This constitutes the continuous mechanism that allows the involved disciplines and functions to coordinate and collaborate on Open Source matters.

The cross-functional role of the Open Source Officer (see pattern Org-3) also needs the Open Source Board as a platform for supporting activities and resolving issues that arise in the daily work and to guide the initiatives towards a higher Open Source maturity for the company.

Considerations

The Open Source Board operates through a recurring meeting and at its initiation, it is important to set the rules, so everyone understands how it works, by defining:

- The responsibility and mandate you want to assign to the Open Source Board. The direction is given by executive management.
- Participation, ideally with a balance between different functions and hierarchical layers. Typically this could be software management, Legal and IPR, Open Source roles (e.g. the Open Source Officer) and potential contribution proposers.
- How decisions are made (needed majority, possible vetoes, decisions at absence etc.)
- Information sharing (minutes of meeting, intranet site, reports etc.)
- If and how to invite external participants when discussing escalations and issues.

The Open Source Board should maintain a tangible understanding of the ongoing Open Source activities in the company by defining, monitoring and communicating a set of Open Source related KPIs. Relevant Open Source KPIs to set up and monitor can be things like:

- Number of contributions (with qualifiers like team, product, contribution type etc.)
- Accepted number of contributions.
- Amount of Open Source code in the product(s) .
- Number of compliance issues (anti KPI).
- Number of community participants on different levels.

Due to the potential business impact (both positive and negative) from introducing Open Source in an company, the Open Source Board should regularly report its activities to the Head of Legal and to a relevant role within business.

Further reading

- C. E. Mols, Krzysztof Wnuk, Johan Linåker , "The Open Source Officer Role – Experiences" OSS 2017: Open Source Systems: Towards Robust Practices pp 55-59

Related patterns

- Org-1 Cooperation with Legal & IPR
- Org-2 Policies, Roles and Authorities
- Org-3 Open Source Officer
- Proc-4 Control Contribution

Open Source Community Culture

> A good starting point is to understand how to empower individuals to act and think freely.

Creating an Open Source community culture is about establishing a de-centralized and highly networking development model where participation and contribution are the key drivers for finding solutions to the common needs. A community culture is instrumental in guiding the Open Source transformation journey by defining a set of shared beliefs that shape the behaviors of the organization.

What it covers

Looking into what drives a community culture, a good starting point is to understand how to empower individuals to act and think freely, which to some degree means de-centralization of the mandate in the classical hierarchy. Key characteristics of a community culture are:

Collaboration. Collaboration is core for a participatory culture since it allows for better idea generation and implementation than can be found in the command and control structure of a hierarchical organization. To encourage collaboration, the community should provide processes and models for work across team boundaries.

Transparency. All technical aspects of the code, design, architecture as well as discussions and decision-making around it, need to be public and open. This secures that everyone can see and act on the same information and establishes a pattern of trust among current and potential participants.

Self-Organization. Open Source communities typically organize according to their needs, which span from flat and extremely interconnected for small ones to semi-rigid hierarchical for the larger ones (like Linux). A development organization that is promoting a community culture should therefore also consider self-organization based on development needs rather than business requirements.

Egalitarianism. Since the contribution model for Open Source is characterized by that anyone is allowed to contribute, it is important to instill that organizational position or belonging of a potential contributor is no limitation. This is not only empowering the individual, but also increasing technological diversity.

Meritocracy. Project direction in an Open Source community is driven by value more than any other requirement. Value permeates the decision structure for contributions with peer reviews, voting and clear feedback loops. Meritocracy with its self-organizing leadership also builds managerial trust.

Apart from the key characteristics, a community culture also entails an expressed vision or policy to strive for active Open Source participation. In this context there is also a need to clarify and share the rules of engagement so that everyone understands the ground rules.

There are many similarities with Agile "culture". Comparing to the 12 principles of Agile there are overlaps in ideas like "harnessing change", "build around motivated people", "attention to technical excellence" and "self-organization". Also, a well-known credo of Open Source communities is "Release early and Release fast" aimed at tightening the feedback loop between

testers or users and developers, and this matches well with the Agile "deliver working software frequently". However, there are also differences – e.g. Agile communication and improvement practices like promoting "face-to-face conversation" and "team reflection" that are not evident in a community culture.

Why it is important

To understand why culture is important, consider the expression "Culture eats strategy for breakfast" (attributed to the management guru Peter Drucker). What it basically says is that any strategy or directive enforced on an organization that is incompatible with its culture will fail. Thus, it can be concluded that introduction of a community culture is required to support the Open Source transformation

Although the definition of organizational culture in itself is very elusive, most people agree that it shapes the behaviors of the workforce based on a set of shared beliefs, whether codified or not. The important question to ask yourself is: What behaviors do we want to encourage and how can we strengthen them? And the answer is: Through culture.

Considerations

Building a culture from scratch (if that is the case) is not an easy task although there is lots of advice around in the management literature. Simply said it is about:

- Owning it – not passing it on to HR or any other department.
- Making sure you are articulating it – by having a joint understanding of the vision and ideas around Open Source and a community culture that is frequently communicated.
- Living it – by living the values, identifying champions to build around and by measuring and rewarding good behaviors.

Outside of instituting cultural values and beliefs, there are a few activities that can help people understand the ground rules of the Open Source culture in your environment, including:

- Set and communicate the company policy around community participation.
- Clarify the local process for how to engage in Open Source communities.
- Give training in community behaviors – the dos and the don'ts expressed as a sort of community interaction guideline.
- For larger organizations, introduction of inner sourcing could be a way to nudge the development teams in the right direction.

In a successfully established community culture it will be completely natural for an engineer to always consider Open Source as an alternative, and to look at any situation from a Make-Buy-Share (pattern Proc-5) perspective.

Further reading

- S. O'Mahony and F. Ferraro, "The Emergence of Governance in an Open Source Community" Academy of Management Journal, Vol. 50, No. 5
- S. Peters and N. Ruff "Participating in Open Source Communities" https://www.linuxfoundation.org/participating-open-source-communities/

Related patterns

- Org-6 Grow Industry Experts
- Proc-5 Make-Buy-Share
- Proc-7 Create and Direct Communities
- Org-8 Collaborative Product Strategy
- Org-9 Self-managed Organization

Grow Industry Experts

> It is central for a company to understand how it can foster and prepare potential community participants for more advanced roles.

To recruit or foster engineers to grow into roles as industry experts is an important step in adapting to Open Source, as it supports the transformation of the internal environment as well as gives a company an ability to direct and govern Open Source projects.

What it covers

A fundamental principle within Open Source is that no company has the power to appoint people into a community governance structure as e.g. contributors or committers – rather this is an earned status based on merit. Thus, it is central for a company to understand how it can foster and prepare potential community participants for the more advanced roles, eventually becoming industry experts. There are a couple of key activities that support this ambition and create the right supportive environment:

Build on culture: Demonstrate the commitment that the company has towards Open Source through a community culture (see pattern Org-5) supported by training, company policies and processes.

Highlight contributions: Show that contributions are important by measuring them with specific KPIs (like: number of submittals, number of accepted contributions etc.) and give recognition to the engineers behind the contribution efforts.

Support individuals: Make sure that engineers have the right support available by 1) mentoring them on the journey from users, through contributors to trusted committers and part of the core team, 2) giving technical training (peer review techniques, community way-of-working etc.) and 3) allowing time dedicated for community activities.

Internal community: Establish a local community with active discussion forums to promote and share the Open Source community understanding and values.

Why it is important

When a company fosters industry experts that are not only valuable for the company itself but also for the community they are participating in, it has essentially gained the ability and trust to direct and govern communities which may eventually develop into ecosystems. In essence, a win-win set-up for all.

Growing experts is not only important for the capability to act more effectively in Open Source communities, it also works as a very strong re-enforcement of the internal community culture and helps to attract further talent.

Considerations

It is important that the company that wishes to foster Open Source engineering skills is clear on the overall direction and recognizes and widely shares contribution statistics regularly. The focus on individual growth can be made even stronger by considering the following:

Incentives: Go even further than just highlighting contributions by making them part of the company incentive schemes. This could range from allowing active engineers to spend more community time, participate in events or even receive monetary compensation. This is not only about rewards, but also to signal recognition and respect.

Performance management: Let community participation and activity be part of the personal development goals. Development career paths can be tailored to community needs by designing the software job grades accordingly. This is about giving purpose.

Build relationships: Relationships on a personal and organizational level are an important aspect of participation in an Open Source community. Attending events and creating an active network should be encouraged for this reason.

Mentoring: Established industry experts should be part of a network within the company, where they can identify and act as mentors for potential future talent.

An alternative that can be considered alongside with internal growth and training is hiring. This can sometimes be the fastest available option if a company is seeking to gain influence in a community. Remember, however, that for retention purposes and general interest from prospective engineers, the hiring company must be able to present a community friendly environment.

Further reading

- J. A. Roberts, Il-H. Hann, and S. A. Slaughter "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects "
- Management Science 200652:7 , 984-999
- https://opensource.com/life/16/1/open-source-skills
- https://opensourceforu.com/2013/06/what-it-takes-to-be-an-open-source-expert/
- https://www.linuxfoundation.org/resources/open-source-guides/improving-your-open-source-development-impact/

Related patterns

- Org-5 Open Source Community Culture
- Proc-7 Create and Direct Communities
- Proc-10 Create and Govern Ecosystems
- Org-11 Authority in Open Source
- Org-8 Collaborative Product Strategy

Open Source Developer Program

It may seem obvious, but it is important that the APIs you are exposing spark external interest – otherwise there will be little or no participance in the Program.

For a company that wishes to improve its external engagement, there is the possibility to start a Developer Program. This can expand the product offering, increase innovation and lead to a better understanding of market needs and challenges.

What it covers

The basis for starting a Developer Program is a business decision around sharing and extending the product software through external collaboration. The audience for the Developer Program may be anything from closed groups of partners to customers that you engage in a win-win set-up where they gain speed, knowledge and depth in using your product, and you gain understanding of their needs, innovative feedback and an expansion of the product offering. It can also be completely open to anyone (any developer) that cares to register.

To start a Developer Program you must be able to share your product through a set of APIs that are made public, which means that you need to work with proper API governance (as partly described in Pattern Prod-1 Modularization and Control APIs). It may seem obvious, but it is important that the APIs you are exposing spark external interest – otherwise there will be little or no participance in the Program.

As part of the Developer Program the company needs to supply an SDK (Software Development Kit) containing the following core elements:

- Tools and resources – to simplify working with expanding your product.
- Documentation and guides.
- Examples – presenting sample code and examples of applications.

You also need to consider having regular technical support that can help with understanding development principles, but also act on issue reports and change requests. To share anything of common interest for the program participants, you need to establish good communication through a web site, blogs or social media channels. To run a Development Program that is regarded as open end listening thus requires a certain effort for the company.

Although there are similarities, the central difference between an Open Source software project and running a Developer Program is that in the latter case it is only the APIs that are exposed externally and that all code (including the definition of the APIs) is fully owned and controlled by the hosting company (You).

Why it is important

There are many good reasons for running a Developer Program:

- Product expansion – letting your partners and customers add to the product via exposed APIs will benefit not only you and themselves but also others.
- Understanding market needs – through external usage of your APIs and the feedback you get, you will learn more about the real market needs.
- Innovation – the addition of external development capability increases the creativity and idea creation around your product.
- Customer satisfaction and loyalty – product expansion leads to better market understanding and higher customer satisfaction with specific adaptations that also gives increased loyalty.

The learnings that a company makes by setting up a Developer Program can also be the seed for starting an Open Source community down the road.

Considerations

Starting a Developer Program is a business decision and needs to be taken considering the current business and its limitations. Thus, it is important to understand how the existing business model for the product can be extended through the program and the public APIs.

As with any software-intensive product, you should use metrics and KPIs that measure the progress and success of the Developer Program. This could be things like number of Program participants, retention, downloads or conversion of the SDK and usage of the APIs.

Create buzz and interest by hosting events, participating in relevant conferences and arranging competitions. Showcase partner and customer solutions through your Program web site or blog (see examples from the very mature and advanced programs at Apple and Google under further reading).

The Developer Program could potentially be a place to spot and recruit talented developers. However, to make sure that you attract the right attendance in your program you will need to invest in using some of your most skilled developers.

Further reading

- Apple Developer. https://developer.apple.com/programs/
- Google Developers. https://developers.google.com/

Related patterns

- Prod-1 Modularization and Control APIs
- Org-6 Grow Industry Experts
- Proc-7 Create and Direct Communities
- Prod-3 Creating a Software Platform

Collaborative Product Strategy

> To gain the advantages of using Open Source, a company needs to accept the reduced control over the product that comes with decision making being done in the communities.

As soon as a company starts to use Open Source software, it will as a consequence no longer have full control of the roadmap for its products. Through more active engagement in communities, the company can to some extent regain control by influencing development. Still, the product strategy of the company needs to be transformed into a collaborative model better adapted to the distributed nature of working with Open Source.

What it covers

Many companies are moving towards a set-up with a high level of de-centralization based on teams with a mandate to take decisions in their own areas of responsibility. This organizational change must also be reflected in how the company is working with the product strategy by transforming it into a Collaborative work model:

- Move from centralized detailed control to high level alignment and shared consciousness.
- Empower development teams so they can contribute to the product strategy.
- Secure that coordination practices are established and there is information transparency both top-down and bottom-up.

Intake of Open Source software basically affects the company in the same way since decisions related to the Open Source components are taken in the communities. Therefore, a collaborative product strategy is very well suited for organizations working with Open Source.

The roles working with a collaborative product strategy in an Open Source environment are typically:

- **Product Management** as a central function shall:
  - Set the long term vision.
  - Drive the Make-Buy-Share strategy.
  - Provide a collaboration framework for the product strategy that the teams can use.
  - When the maturity level has increased – understand and utilize crowd-based requirements engineering (see pattern Proc-9 Crowd-based Requirements).
- **Teams and individuals** working within Open Source shall:
  - Share information on what is happening in the community.
  - Identify opportunities (e.g. through external scouting activities or crowd-based requirements on a more advanced level, see pattern Proc-9).

For the different Open Source components that the company is using, there is a need to set a strategy for its level of engagement. This is partly covered in the patterns Proc-5 Make-Buy-Share and Proc-4 Control Contribution. Minimum is to establish an internal "owner" that keeps track of the component development, updates and potential security patches.

Why it is important

To gain the advantages of using Open Source, a company needs to accept the reduced control over the product that comes with decision making for Open Source components being done in the communities. Consequently, the product strategy has to be adapted to a collaborative model with both top-down and bottom-up characteristics, unlocking more of the opportunities offered by Open Source. The resulting product management way of working is similar to scaled agile systems.

Considerations

Looking at the Open Source transformation journey overall, the fact that the company needs to sacrifice its full control over the product is one of the mind-set transformations to manage. The perceived disadvantage of losing control needs to be contrasted with all the advantages of Open Source in general and the power of a collaborative model in particular. This organizational transformation needs to be addressed early on in the Open Source journey by explaining the way forward and how this is, after all, much better.

In line with the above, the people working within product management will most likely need coaching and training to understand how to gain the most benefits out of a collaborative model, stop trying to define all the details of the product roadmap themselves and starting to work more through alignment, long term visions and Open Source strategies.

Understanding commoditization in the software industry is one of the critical factors that can lead to an acceptance of adopting a collaborative product strategy. Companies that nonetheless still want full control over the code and are willing to pay the price for it, will need a strong business incentive as justification.

Making a successful product strategy in an Open Source environment requires understanding of many aspects of Open Source (in this booklet described across several patterns) and how it affects the company. This includes things like Make-Buy-Share strategies, component contribution strategies and community culture – and the Open Source Board that is driving the Open Source journey in the organization.

Further reading

- M. DeHaan “6 steps to perfecting an open source product strategy” https://opensource.com/article/17/9/demystifying-open-source-product-strategy
- D. Neary “Crafting an Open Source Product Strategy”, Open Source Community of Redhat https://community.redhat.com/blog/2018/04/crafting-an-open-source-product-strategy/

Related patterns

- Org-4 Open Source Board
- Org-5 Open Source Community Culture
- Proc-5 Make-Buy-Share
- Org-6 Grow Industry Experts
- Proc-4 Control Contribution
- Proc-9 Crowd-Based Requirements
- Proc-7 Create and Direct Communities

Self-managed Organization

> The company leadership provides a clear direction that is shared and understood in the sense of a common consciousness. They do not lead by command in the traditional way, but make sure that all teams have the means and the mandate to do what is needed.

A company that is based on self-managed teams supported by a visionary leadership has a much higher probability of flourishing in the complex, collaborative and dynamic environments of the Open Source world. This type of organization will be able to identify and harvest new business opportunities and use Open Source communities and ecosystems to realize them.

What it covers

Changing the organization in the company towards being flatter, self-managed and bottom-up oriented with a coaching rather than commanding leadership style can be a large and complex effort. It involves an overhaul of the company culture as well as transformation of roles and responsibilities of leadership, teams and the individuals.

Culture is an important fundament for any company and especially so for a self-managed organization where culture is a part of the framework that makes it all tick. As expressed in the pattern Org-5 Open Source Community Culture, the key characteristics are transparency (information is openly shared), egalitarianism (anyone can contribute regardless of position or role) and meritocracy (self-organized leadership based on value and trust).

The company leadership provides a clear direction that is shared and understood in the sense of a common consciousness. They do not lead by command in the traditional way, but make sure that all teams have the means and the mandate to do what is needed. Cross-organizational concepts like establishing a company vision, defining business models and setting ecosystem strategies are still a responsibility for the leaders. For the product strategy, this can be seen as a further evolution of the pattern Org-8 Collaborative Product Strategy.

The fundamental organizational unit is the team, consisting of different roles to make it as autonomous as possible. It is self-managed based on the shared vision, but maintains a close and regular communication with other teams to create understanding of their context. New methods and tools are decided on and adopted at the discretion of the team, but cross-functional alignment is expected. Decision making is always done at the lowest possible level, which is often the team. This way, ad-hoc business opportunities can be harvested and driven bottom-up.

Although everyone is part of and contributing to a specific team, the individual in the self-managed organization has the privilege to contribute to work done in any team, to swap teams if wanted or needed and to explore opportunities that emerge. In all this freedom, there is on the other hand also an expectation to step in for others when needed – even if this may be outside of the regular area of responsibility. This is in direct accord with the ambition to grow competence and recognition described in pattern Org-6 Grow Industry Experts.

Why it is important

There are several reasons for why a self-managed organization is advantageous. Due to the similarities with the set-up of Open Source communities, it

will be considerably better at interacting with these. It will also be well adapted to harvesting Open Source business opportunities. Apart from the specific Open Source oriented advantages, it will also enjoy generic benefits like high speed decision making, a natural adaptiveness to change, enabling innovation and empowerment of employees.

Considerations

For large organizations, the introduction to a self-managed set-up is very challenging since it involves changing culture, expectations and behaviors and involves lots of communication, training, involvement, short and long-term goals and visions. As in all such transformations, it is better to break down the change in reasonable steps than to do it all at once. The important thing is to establish a shared understanding of the wanted state so the self-managed teams can interact with remaining pockets of traditional style organization in a good way.

Some suggestions on how to introduce a self-managed organization:

- Start with development teams and gradually increase their responsibility to include business aspects.
- Select teams that cover a specific product or a well-defined part of a product.
- Then make sure that it covers all of the software organization.
- Finally, introduce it across the whole company.

Companies that have embarked on the transition to Agile will recognize many of the principles, although the self-managed organization is a wider change where team and individual freedom is larger and the perspective on business is broader.

The organizational change will most likely happen in parallel with the change to become a software-centric service oriented company (see e.g. patterns Prod-4 Own Service Offerings, Prod-6 Service-based Business and Org-10 Directed by Business Aspects). This tends to be something that makes the change simpler and more natural since it goes hand in hand with the exploratory and business-oriented way of working in the teams.

Further reading

- Mruzik & Peters, Smart Business, 12/1 2017, http://www.sbnonline.com/article/look-self-managed-corporate-structure/
- Frederic Laloux, Reinventing Organizations, 2014-02-20, Laoux, ISBN: 9782960133509
- W. Ke and P. Zhang, “Effects of Empowerment on Performance in Open-Source Software Projects”, IEEE Transactions on Engineering Management, vol. 58, no. 2, pp. 334-346, May 2011.
- H. Holmström Olsson, J. Bosch, “No more bosses?: A multi-case study on the emerging use of non-hierarchical principles in large-scale software development” , International Conference on Product-Focused Software Process Improvement PROFES 2016: Product-Focused Software Process Improvement pp. 86-10.
- W. Aghina, A. De Smet, G. Lackey, M. Lurie, M. Murarka “The five trademarks of agile organizations”, McKinsey https://www.mckinsey.com/business-functions/organization/our-insights/the-five-trademarks-of-agile-organizations”

Related patterns

- Org-5 Open Source Community Culture
- Org-8 Collaborative Product Strategy
- Org-6 Grow Industry Experts
- Org-10 Directed by Business Aspects

Directed by business aspects

> Open Source engagement makes it simpler for the company to continuously deliver additional value to the customers to maintain price levels.

This pattern describes how a company can become directed by business aspects of Open Source. This primarily includes identifying and harvesting new business opportunities from various forms of Open Source engagement. In addition, the company must also understand and apply new business models enabled by Open Source and explore new business areas.

What it covers

This pattern covers two main elements. The first one is about understanding and utilizing new types of business models that are made possible through Open Source involvement. The second element describes the organizational capabilities needed to identify and harvest business opportunities.

The new types of business models include:

- Extended business models, where a company monetizes additional services offered together with the software rather than the software itself (e.g. support services, freemium, add-ons, supplementary training, tools, dual-licensing).
- Indirect business models, where software is included into the price of a hardware or service offering (e.g. selling hardware at premium price and software is included "free of charge" or selling services and Open Source software enables those services to be run).
- Asymmetric business models, where revenue is based on the effects of running software and collecting data. Asymmetric business models are important for data monetization strategies that often involve sharing an Open Source solution for free and creating revenue streams from customer data inserted into the solution (e.g. provide Android for free and get revenue from ads or monetize user data through direct and indirect marketing activities).

Ensuring capabilities to identify and harvest business opportunities that Open Source participation brings, requires an organization that is self-managed and empowered (as described in pattern Org-9 Self-Managed Organization).

Opportunities that arise are often market disruptive (like entering new markets with an Open Source platform or create a new Open Source ecosystem to challenge proprietary solutions in a given market segment – examples can be found in the Further Reading material). Working with Open Source solutions on this level requires organizational maturity in establishing Open Source structures and delivering software that sparks an interest in the Open Source communities.

Why it is important

It is difficult to sell software as a product in a "traditional way". Open Source has changed the pricing models and ways of monetizing your software development efforts. There is a set of good reasons for a company to develop

the capabilities required for leveraging the business potential residing in Open Source:

1. A company that has the capability to work with Open Source on a high maturity level will also be able to harvests business opportunities from Open Source involvement. It understands and can exploit revenue streams that are based on extended, asymmetric and indirect business models.
2. Open Source engagement makes it simpler for the company to continuously deliver additional value to the customers to maintain price levels (e.g. services, data analytics or other forms of customer business support). Without additional value the customers will pressure for price optimizations that further limits cost of development.
3. Open Source participation lowers entry barriers significantly and enables new business areas to be pursued.

Considerations

There may be organizational resistance to simultaneously handle several business models and move between them when appropriate. Thus, a significant change management effort may be needed to get to a company that is driven by business aspects of Open Source.

Another important aspect is that this pattern requires several other patterns as prerequisites, e.g. pattern Org-6 Grow Industry Experts. It is also important to share an understanding of the business side of Open Source participation with the engineers, to help direct development efforts towards value creation and capturing relevant functionality.

Finally, what engineers believe are good features may not be appreciated by the market and the customers. Not all ideas will result in creating new Open Source communities. Therefore, establishing evaluation mechanisms to select ideas with the right potential is important to avoid creating "solutions that are looking for problems". Sometimes this means that a few engineer darlings need to be killed.

Further reading

- Lerner, J. and Tirole, J. (2002), "Some Simple Economics of Open Source". The Journal of Industrial Economics, 50: pp. 197-234.
- M. Svensson, M. Agarwal, S. Terrill, K. Wallinn, "Open, intelligent and model-driven: evolving OSS", https://www.ericsson.com/en/ericsson-technology-review/archive/2018/open-intelligent-and-model-driven-evolving-oss
- B. Fitzgerald, "The Transformation of Open Source Software", MIS Quarterly Vol. 30, No. 3 (Sep., 2006), pp. 587-598
- K. Sandeep, "An Analysis of Open Source Business Models. Making Sense of the Bazaar: Perspectives on Open Source and Free Software" , MIT Press

Related patterns

- Org-11 Authority in Open Source
- Org-9 Self-managed Organization
- Org-6 Grow Industry Experts
- Proc-10 Create and Govern Ecosystems
- Proc-8 Industry-wide Collaborations

Authority in Open Source

> The large foundations increasingly define what ends up as de facto world standards based on Open Source. Becoming an authority in Open Source gives access, leverage and influence into the foundations.

Larger, global Open Source organizations, often managed as foundations, increasingly define what becomes de-facto world standards. To gain access and take leadership in those, thus being able to influence and drive market-shifting initiatives, a company must strive for and become a leading Authority in Open Source.

What it covers

Already in the early days of the Free and Open Source software movement it came naturally to manage development in the non-profit organizational form as a foundation. Richard Stallman founded the Free Software Foundation as early as 1985, the Linux Foundation can trace its origins to Linux International in 1993 and the Apache Foundation to 1999. As these foundations grew massively in size over the years, their specific software technology domains grew as well to the point today that they host a vast variety of Open Source project of which many are engaged well beyond the original intent.

The foundations have become so dominant in software technology development that they today by far overreach the abilities of the traditional bodies for industry collaboration, the standards institutes. Thus, the foundations have become able to establish their technology with a near universal uptake and following, to the extent that the technologies are perceived as de-facto standards. Interesting enough, it has lately been observed that some standardization bodies, e.g. the European Telecommunications Standard Institute, are considering adopting Open Source practices for their standardization work.

The Authority in Open Source pattern is about a company having **sufficient organizational capacity and recognition** to drive and direct the marketplace through industry-wide collaborations as found in the foundations.

Capacity as an Authority in Open Source is generally required to gain a seat in a Foundation's governance board. A governance board is seldom allowed to interfere with the development work within a foundation's Open Source projects. However, as the governance board has the responsibility for securing the overall success of a foundation, it decides on which projects to run or close. Hence, having a seat in a foundation's governance board supports the viability of your Open Source project, paving the road for it to become a de-facto standard, and in the long run securing its prospect as a new market offering. This implies that the **capacity** will rely heavily of on a generous availability of a well-seasoned staff with both software industry expertise as well as managerial skills in collaborative engagements.

Recognition is generally achieved by sharing comprehensive knowledge and expertise on all the different aspects of Open Source such as:

- Experiences on different governance models for communities.
- Mastering different Open Source-based business models.
- Access to world renowned legal expertise on Open Source.
- Extensive research and education in the area of Open Source.

Note that a considerable **organizational capacity** in both staffing and communications is required to achieve the above. The tools of communications includes activities such as workshops, hackathons, conference talks, a heavy

presence on the web, community management, etc., all aiming to maximize the knowledge exchange which in the end should lead to a **recognition** as an Authority in Open Source. As such, a company would likely find that instead of chasing after business opportunities, they would come to the company. This as an Authority in Open Source obviously is open for the widest possible industry collaborations while it also is accepting to take on a leading role in the marketplace.

Why it is important

A major trend the latest years is that Open Source communities are merging into a handful larger Open Source organizations, often managed as foundations, thus covering a wider area of interest. An example is the Linux Foundation, which has lately emerged as the Open Source organization *par excellence*.

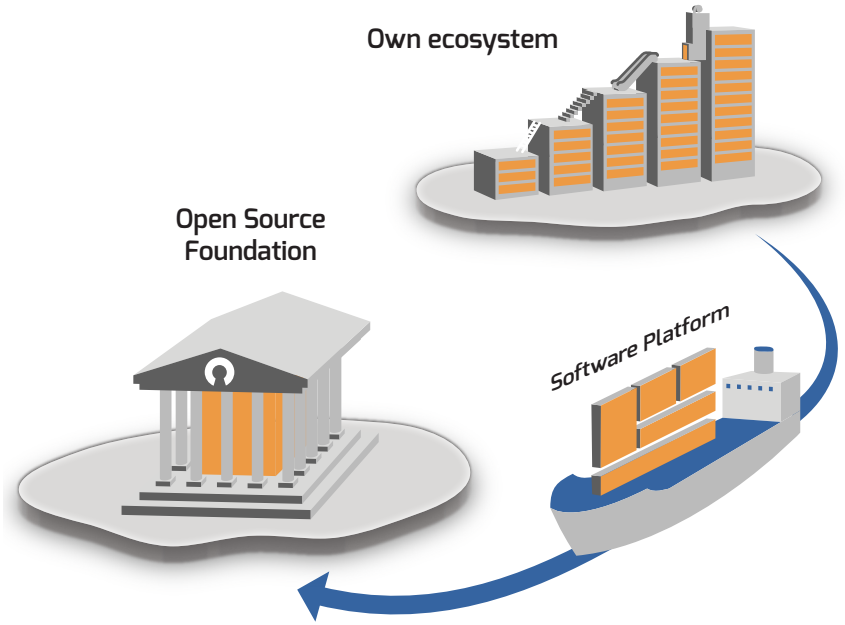
The large foundations increasingly define what ends up as de facto world standards based on Open Source. Becoming an authority in Open Source gives access, leverage and influence into the foundations, thus having a say of some weight on what will become industry-wide used standards.

Considerations

It's not for everyone to be recognized as an Authority in Open Source as it **requires considerable organizational capabilities**, in par with creating and maintaining an own ecosystem. As recognition is achieved by sharing expertise, it's key to have a **comprehensive participation** in the work of larger Open Source communities, supplemented by **extensive communication**. This indicates that it will be too challenging for smaller companies.

In order to lower the overall organizational burdens, it could also be considered to **move the management of an Open Source platform** of an own orchestrated ecosystem to a foundation. As this also enhances the opportunity of the platform becoming recognized as a de-facto standard, such a move would likely secure success in the market (see picture).

An example of this is Kubernetes, an Open Source system for the management of containerized applications. Originally designed by Google, it is now maintained by the Cloud Native Computing Foundation, which is part of the Linux Foundation. Since its release in June 2015, Kubernetes has established itself as one of the hottest available cloud technologies and have quickly attracted a massive attendance from the industry.



A quite common alternative to launch an Open Source platform in an existing foundation is to create a specific foundation for the platform, while untangling yourself from full stewardship. Two good examples: IBM with their original Eclipse IDE (Integrated Development Environment) that became the Eclipse Foundation, and likewise what server company Rackspace pursued with their open OpenStack cloud platform, as it became planted into the OpenStack Foundation. Both serve as good examples of how to create backing from the industry and establish de-facto standardization in their corresponding areas.

Further reading

- The Cloud Native Computing Foundation: https://www.cncf.io/
- The Linux Foundation: https://www.linuxfoundation.org/

Related patterns

- Org-7 Developer Program
- Org-5 Open Source Community Culture
- Org-6 Grow Industry Experts
- Org-10 Directed by Business Aspects
- Proc-7 Create and Direct Communities
- Proc-10 Create and Govern Ecosystems

Product Patterns

Modularization and Control APIs

> Open the opportunity to plug in Open Source solutions into your codebase and allow mixing of Open Source and proprietary code.

Modularization is a generic software engineering practice that supports decomposition of large systems and enables a structured way of working with Open Source.

What it covers

Modularization is a commonly known software practice that is especially relevant for large, complex software systems. Modularization implies that:

- Code should be separated into logically independent modules, i.e. separated based on distinct tasks that the code performs.
- All internal details of a module should be hidden behind a public interface (API, Application Programming Interface).

With properly modularized code you can expect to gain these advantages:

- Code will be easier to understand and troubleshoot.
- Testing will be less painful (and automated testing is facilitated.)
- It will be simpler to reuse and re-factor the software.

So, what is the connection to Open Source? First of all, it will open the opportunity to plug in Open Source solutions into your codebase and allow mixing of Open Source and proprietary code. This will certainly be possible even in a monolithic structure, but it will be at a much higher cost both for introduction and maintenance (and make it much more difficult to control compliance). Second, it will enable Open Source code under incompatible licenses (that can be isolated in different modules). Third, in a modularized architecture it will be possible to create your own Open Source projects from suitable components in the system.

With modularization there also follows organization-wide or public APIs, and thus the need to maintain and govern these over time. In particular for a large company there needs to be policies on how to publish, promote and maintain APIs. It is easy to understand unwanted impacts of weak design or changes to APIs that are publicly available, but even for internal APIs it is important to keep a good level of discipline (a point that has been extremely clearly expressed by e.g. Amazon as can be seen under Further Reading).

Why it is important

There are several positive effects on Open Source from doing modularization:

- It makes it simpler to integrate Open Source solutions into existing software.
- It facilitates a more complex usage of Open Source like using many different Open Source project solutions.
- It's a pre-requisite for building platforms and eco-systems to increase competitiveness in the long run.
- Finally, it makes it possible to take the next step on the Open Source journey, i.e. to contribute your own code to Open Source projects or even create Open Source projects.

Considerations

Since modularization is a fairly common practice within software development, there are many helpful tips on what to think about, for instance:

- Aim for Cohesion – which basically means that a well-designed module should cover a specific job or task that makes sense and not just be a collection of logically unrelated functions.
- Low Coupling – the need for data that is kept in other modules should be as low as possible, i.e. effective de-coupling.
- Information hiding – the inner logics of a module should be hidden.

Make sure that your APIs are regularly monitored to check that they are used as intended or if the traffic is unusually high or low suggesting that they need to be changed or can be removed.

For small companies it may seem like a hefty task to do full API governance and maintenance, but there are many tools to use that can lessen the burden considerably.

Further reading

- http://apievangelist.com/2012/01/12/the-secret-to-amazons-success-internal-apis/
- K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen. "The structure and value of modularity in software design." SIGSOFT Softw. Eng. Notes 26, 5 (September 2001), 99-108.

Related patterns

- Proc-1 Control Compliance
- Proc-2 Control Intake
- Prod-2 Code Management
- Proc-6 Frequent Releases
- Proc-4 Control Contribution
- Proc-5 Make-Buy-Share
- Proc-7 Create and Direct Communities
- Org-7 Developer Program

Code Management

> By having control of where proprietary, third party and Open Source reside in the system, the organization will be able to work with Open Source in an effective and professional way.

Code Management describes what you need to know about your codebase in order to do the right things and take the right decisions in an environment that includes code from different sources, like Open Source software. Component level control enables effective and professional governance of Open Source, as described in other patterns.

What it covers

The first step of Code Management is to produce a full inventory of all your components including information on where and what Open Source is used. Apart from the issues around defining and counting components in what may be a complex system, this could seem like a straightforward task given that the company is in control of its Open Source usage – but consider that there are many ways for Open Source to enter software "under the radar". These include unsolicited developer downloads, code reuse, inclusion of commercial applications, third party code and outsourced development to name a few.

When doing the inventory, there will be a need for metadata information like origin project, version number, license type, dependencies, technology involved and implemented functionality. The resulting component catalog is not only applicable for Open Source, but can be used generally for things like setting up the software bill of materials (SW BOM) for your products or tracking of third party components.

Once the catalog has been established, it is equally vital to keep it updated and relevant over time, not least given all the changes a codebase usually undergoes. The information on Open Source components contained in the catalog will also support other patterns like compliance and product strategy (see patterns Proc-4 Control Compliance and Org-8 Collaborative Product Strategy).

Why it is important

Code Management is mainly an enabling pattern that prescribes the compilation of a component catalog for the codebase of a company. By having control of where proprietary, third party and Open Source reside in the system, the company will be able to work with Open Source in an effective and professional way as described in some of the other patterns.

The inventory of Open Source will also be helpful when maintaining products that are released to the market. New releases of Open Source components may be essential to update due to e.g. major issue fixes or newly discovered security vulnerabilities. Considering that Open Source is likely to be part of the codebase even if it was not put there deliberately by you, it is easy to understand that the component catalogue is an indispensable tool.

Considerations

A modular structure of the software (as described in pattern Prod-1 Modularization and Control APIs) will considerably simplify Code Management.

As the codebase in the company grows, it will become increasingly difficult to maintain a full view of all the components in the system. At that point it can be wise to consider a tool. There exists a large variety of commercial and Open Source systems that can help out with Open Source management ranging from doing inventories (as mentioned in this pattern) to governance and compliance. Such tools will generally also support Code Management for proprietary and third party components and tool selection should take all mentioned factors into account.

Although Open Source favors distributed decision making and organizational structures, Code Management is a practice that has to be governed centrally. If you want to secure a complete documentation of the system, there should only be one way of doing it.

Further reading

- I. Haddad “Using Open Source” https://www.linuxfoundation.org/using-open-source-code/

Related patterns

- Prod-1 Modularization and Control APIs
- Proc-1 Control Compliance
- Proc-2 Control Intake
- Proc-4 Control Contribution
- Proc-5 Make-Buy-Share
- Org-8 Collaborative Product Strategy

Creating a Software Platform

> All code that is used in cross customer implementations should be in the platform.

Creating a software platform establishes cost-effective code reuse mechanisms between projects, increases code quality and stabilizes interfaces. It also enables a company to develop complementary services based on the platform.

What it covers

A software platform offers code reuse between projects, gives lower maintenance costs and improves code quality. The architectural principle behind creating a software platform is to divide the complete system into reusable parts (the actual platform) and project specific parts (variants). (This is according to the Software Product Line principle that can be found under Further Reading.) The development environment and tools are shared between projects and variant development is done on top of established and stable platform interfaces. The obvious advantage of creating a platform is that it increases software development efficiency and effectiveness by reusing rather than developing most of the code in each project. However, it also makes it simpler to add additional services on top of the platform.

To ensure evolution and maintenance of the platform, each software project that is based on a platform needs to include the following steps:

1. Get the latest version of the platform code and configuration files that describe possible variants and versions.
2. Plan for additional development of features.
3. Decide if the developed features are going to be integrated in the platform or not (as much code as possible should be reused).
4. Integrate selected reusable features with the platform code and update the configurations.

Why it is important

The main benefits for organization from building a software platform include:

- Creating a software platform increases code reuse between projects – since new project are based on previous development activities. Thus, development time and maintenance costs decrease, and code quality increases, as code improvements propagate to all future products
- Complementary services can be offered – the software platform ensures that services are possible to deliver to all products based on the platform. Development resources can be dedicated to service development rather than platform maintenance
- Code quality and interfaces – reuse increases code quality as potential errors are detected, fixed and integrated into the platform rather than a single product. Clear and stable interfaces and APIs are the prerequisite for reaching stable high-quality platforms.

Considerations

Raising code quality levels and providing clear and stable interfaces are the key elements in building a software platform. Moreover, modularity is necessary to enable cross-project reuse (see pattern Prod-1 Modularization and Control APIs).

Creating a platform must be synchronized with increased transparency of the requirements engineering processes and sharing strategic plans for the platform with the projects that reuse the code.

When a company has created a platform, it should consider if it can be released as Open Source since this gives additional opportunities in terms of e.g. shared development and extended innovation (see pattern Prod-5 Open Source Driven Platform Innovation).

Platforms should be created based on reuse principles, meaning that all code that is used cross customer implementations should be in the platform. This often implies that platform code represents non-competitive parts of the offering. The risk of losing valuable IPR is therefore low for these parts and they can be shared openly with the communities.

Further reading

- Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- Sony Mobiles Open platform https://developer.sony.com/develop/open-devices/ as an example.

Related patterns

- Prod-1 Modularization and Control APIs
- Prod-5 Open Source Driven Platform Innovation
- Prod-6 Service-based Business

Own Service Offerings

> Customers have also come to expect services being added to more advanced products, thus adding value throughout their lifetime.

When a company starts to offer services as a complement to its products, it is in essence extending its product offerings and opening up for new revenue streams. Through the services the company also has an opportunity to create a closer relation to its customers and partners.

What it covers

Servitization is a transformation journey where companies (think: industrials) develop capabilities needed to provide services and solutions that complement and extend the traditional product offering. In this transformation, a shift happens from developing and selling a product to the customer, to a system where the company evolves its capabilities and processes towards creating mutual value (for the company as well as the customers) through services. This pattern describes the first step on the journey.

To give a better idea of what the first level of services could be, consider services like scheduled maintenance, technical helpdesk, repairs, product overhaul, installation, operator training and certification, condition monitoring and in-field service. Hence, the customer still owns the product, but services that go beyond the initial ("one off") sale are also offered and they are typically value-adding when being used (over the lifetime of the product).

Servitization is a generic concept, but it provides a good mechanism to monetize usage of Open Source. A company that is starting to work with services also has an opportunity to open up APIs for partners and developers. If this is done through e.g. a developer program (see pattern Org-7), there is an opportunity to extending the service portfolio through a collaboration with partners and customers. This is the starting point of recognizing that the product is a platform from which services can be developed.

Why it is important

Extending the product offering by offering services is basically a way of gaining a competitive advantage. This can be forced on a company struggling with low cost competition to its product, but regardless if this is the situation there are certainly also good financial incentives for developing services, since they complement the "one off" sale of the product with new revenue streams. Furthermore, using Open Source removes the question of value of the software itself and instead puts focus on what the services offer and what revenue can be created from them.

Apart from increasing competitiveness and creating new revenue, customers have also come to expect services being added to more advanced products, thus adding value throughout their lifetime. For companies that offer services through a cloud implementation, there is also the advantage of simpler version control.

Considerations

One of the largest challenges with servitization lies in the change from a traditional product view to a service culture. Understanding that the product can be a platform to deliver services is a big step that impacts a large part of the company. Making the change incremental by introducing the type of first-level services mentioned above is often a good idea, but it still involves dramatic changes to e.g. marketing, sales and KPIs.

Both technical and capability factors need to be considered when starting with services including things like security, privacy, usability, robustness (should there be 24/7 availability, for instance) and performance. These are things that have probably not been needed to manage before.

Collecting, storing and analyzing data in a data driven way is an opportunity that becomes more available as the interaction with the customer is enhanced with servicese. This requires the company to learn how to do it and understand how this can offer market and customer insights that will improve the portfolio.

Further reading

- Andy Neely, 30/11 2013, http://andyneely.blogspot.se/2013/11/what-is-servitization.html

Related patterns

- Prod-1 Modularization and Control APIs
- Org-7 Developer Program
- Prod-3 Creating a Software Platform
- Prod-6 Service-based Business

Open Source Driven Platform Innovation

> If successfully managed the platform may become a de facto industry standard.

Open Source Driven Platform Innovation is about releasing your platform as Open Source software to enable two kinds of innovation: 1) additions to the platform and quality improvement suggestions from the Open Source community and 2) extracting products from Open Source software platforms.

What it covers

A company that creates a software platform can decide to release this platform as Open Source software meaning that the platform code and its interfaces are shared with an Open Source community. The goal for the company is to attract external innovation by allowing new functionality and additions to be developed by others and aim for the Open Source platform to become a de facto standard. Enabling external involvement (also called harvesting the "Open Innovation" model, see Further Reading material) significantly speeds up development and frees up internal resources for other activities.

The company that owns platform becomes the platform leader and manages governance and other activities needed to launch the Open Source platform:

- Platform participation rules and Open Source licenses are established.
- Contribution processes are introduced.
- Communication and discussion channels are established and made available for interested stakeholders and developers.
- Relevant information is published in a transparent way (e.g. the platform roadmap, development rules and processes)
- Marketing activities should follow to attract developers or other companies to join the platform and create products based on the platform.

Releasing the platform as Open Source ideally shifts innovation towards happening externally (in the community) rather than internally. Based on how and by whom the Open Source platform is used and innovated on, this gives the company a possibility to understand market needs and extract products either from the platform itself or as new functionality in separate products.

Finally, when the Open Source platform receives wider adoption, the platform leader can utilize the contact network of the organizations using the platform to establish industry-wide collaborations or offer additional products (or services and consulting around the platform). (Also see pattern Proc-8 Industry-wide Collaborations)

Why it is important

Launching a software platform as Open Source brings several benefits to the company. If it is successfully managed, the platform may become a de facto industry standard, which creates a competitive advantage compared to other companies. Even if this potential is not fully reached, a number of benefits can still be achieved:

1. It further increases code quality since Open Source users and developers act as additional testers and quality assurance experts - suggesting improvements to the platform.
2. It broadens the usage of the platform – other companies find new use cases and create new products based on the platform.

3. It brings opportunities to co-create value with companies that extract products from the platform and form alliances in new markets or business areas.

4. It creates a network of organizations using the Open Source platform. This could trigger industry-wide collaborations.

Considerations

Releasing a software platform as Open Source is clearly a business decision. There should be a good rationale behind the decision, a plan for how platform governance should be set up and a clear strategy for how to leverage on the platform when it is Open Source.

Good Open Source candidates are software platforms that do not bring significant revenue (require extension with services or additional components), or platforms that risk becoming commoditized but can still spark external interest. Thus, it is critical to understand when and how the market for the platform matures and when the cost of ownership becomes bigger than potential revenues.

Releasing a software platform as Open Source requires establishing stable interfaces and communication and coordination mechanisms. It is also important to remove the participation barriers and create educational material for developers interested in joining the Open Source platform, e.g. online courses or wiki-based instructions how to start developing additional functionality on the top of the platform. Attracting developers is critical when the Open Source platform is announced, and this should be supported by public events (like developer days, sessions at industry conferences).

Management taking the release decision need to ensure that the necessary technical infrastructure is in place (for online discussions and distributed development), that the "platform launch" event is planned and coordinated and that publicity activities follow. Finally, long-term commitment and dedicated moderators and facilitators are needed to keep activity in the community alive and to keep external developers' motivation. This requires dedicated industry experts to govern the platform development.

Further reading

- H. Chesbrough, W. Vanhaverbeke, and J. West, Eds., "New Frontiers in Open Innovation". Oxford University Press, Nov. 2014.
- H. Munir, J. Linaaker, K. Wnuk, P. Runeson, and B. Regnell, "Open innovation using open source tools: a case study at Sony Mobile," Empirical Software Engineering, pp. 1–38, 2017.
- Khurum, M., Gorschek, T. and Wilson, M. (2013), "The software value map — an exhaustive collection of value aspects for the development of software intensive products". J. Softw. Evol. and Proc., 25: 711–741.
- J. Linåker, H. Munir, K. Wnuk, C.E. Mols, "Motivating the contributions: An Open Innovation perspective on what to share as Open Source Software", In Journal of Systems and Software, Volume 135, 2018, Pages 17-36.
- I. Haddad, F. Benard "Good and Bad Reasons to Open Source Your Software How do you measure up? " http://www.ibrahimatlinux.com/uploads/6/3/9/7/6397792/05.pdf
- J. Bosch "From Software Product Lines to Software Ecosystems", 13th International Software Product Line Conference (SPLC 2009) August 24-28, 2009, San Francisco, CA.

Related patterns

- Prod-3 Creating a Software Platform
- Proc-8 Industry-wide Collaborations
- Proc-9 Crowd-based Requirements Engineering
- Proc-10 Create and Govern Ecosystems
- Org-10 Directed by Business Aspects

Service-based Business

> On the highest level of servitization, the product business is completely replaced by a service infrastructure. The infrastructure is implemented as a platform.

As the level of servitization evolves and increases in the company, there may still be a product, but this will no longer be what is actually monetized. By marketing and selling services, the business has transcended the barriers surrounding products and can enjoy a win-win-win situation for all – the company, its partners and its customers.

What it covers

When the company continues on its servitization journey, it will reach a point where its business is completely based on services. There can still be a product, but this is owned by the service provider or a partner. Examples of services offered on this level can be revenue-through-use contracts or rental agreements.

The shift to services constitutes a paradigm shift for how business is viewed in the company. The interests of the service supplier, the distributors or partners and the customers are joined in a win-win-win relation where the financial incentives are better tuned to actual needs and opportunities. Rolls Royce aircraft engine CorporateCare system ("Power by the Hour") serves as a good example where the customers (the airlines) get hassle-free flying hours and Rolls Royce can optimize the complete chain of production, maintenance and exchange.

On the highest level of servitization, the product business is completely replaced by a service infrastructure. The infrastructure is implemented as a platform, and often the service(s) and the platform are part of an ecosystem. Examples of companies offering services on this level are Netflix, the entertainment company that transformed from a DVD by mail distributor to a streaming media service, and RackSpace, the cloud computing company building much of its success on OpenStack – a platform for cloud computing – which they released as Open Source software in 2010.

To further secure success of the platform that is used as a base for services, partners and customers are often invited to collaborate. This is also a good reason for offering the platform as Open Source software (as opposed to a proprietary system) since it builds trust towards all participants. Services, platform, tools, a market place and governance together make up an ecosystem (see pattern Proc-10 Create and Govern Ecosystems to understand how to set this up).

Why it is important

Servitization and Open Source software is a perfect match since business through services can be seen as the main mechanism to get paid for working with Open Source on an advanced level. There are, of course, other good reasons for moving to services, like:

- The cost structure is better for the customer (they pay for what they need and the payment is distributed over time).
- If there is hardware involved, the utilization and reuse of it can be optimized.
- Company resources are shared across customers.
- A service based business scales better, investments in new features or capability are moderate and adaption to changing market needs are simpler.

Considerations

Since the business models need to change when the company goes through a servitization transformation, it becomes very important to understand the different options for revenue creation. This is described in the pattern Org-10 Directed by Business Aspects.

On the highest level of servitization where there is no product business, it may be challenging to get a paying customer base. The difficulties of selling fully service based solutions are often underestimated. In preparation, it is important for the company to answer questions like "Is the market ready to accept the new value proposition?" and "Is the organization ready to deliver it?"

Data Driven methodologies, where data feedback loops are used to direct development, become increasingly important for fast market adaptation and to increase customer satisfaction. To secure success of such practices, all the basic capabilities (patterns) of the organization, process and product types need to be in place.

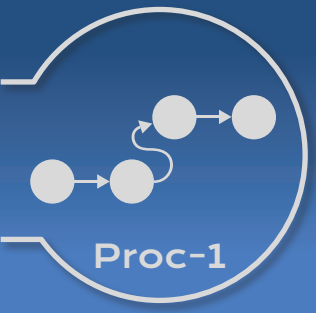
Further reading

- M. Turner, D. Budgen and P. Brereton, "Turning software into a service," in Computer, vol. 36, no. 10, pp. 38-44, Oct. 2003.
- World Finance, 8/3 2016, https://www.worldfinance.com/markets/rolls-royce-is-driving-the-progress-of-the-business-aviation-market
- https://www.rackspace.com/

Related patterns

- Proc-10 Create and Govern Ecosystems
- Org-10 Directed by Business Aspects of Open Source
- Prod-4 Own Service Offerings
- Prod-3 Creating a Software Platform
- Prod-5 Open Source Driven Platform Innovation

Process Patterns



Control compliance

> The Open Source licenses have to be unconditionally followed if the company distributes Open Source licensed software.

Open Source software developers must follow the licensing conditions. This means that the company must implement appropriate routines and tools to control compliance to Open Source licenses.

What it covers

The Control Compliance pattern is about implementing the appropriate routines and tools in a company to control compliance to Open Source licenses. This involves defining and communicating which Open Source licenses your company should and should not use and how to manage those. License types are often grouped according to the following:

- Blacklisted licenses – that will not be approved.
- Whitelisted licenses – that are generally OK to use, given the right context (like given that there is no risk for incompatible licenses).
- Pre-approved Open Source systems – that supply basic functionality and have been approved for usage across the product (like Google Android in mobile devices).

A solution to track what Open Source components that are included must be implemented to support the process. The objective is to have an inventory list of all open source components and their respective licenses (also see pattern Prod-2 Code Management).

The Open Source licenses have to be unconditionally followed if the company distributes Open Source licensed software. This covers any kind of arrangement in which the software leaves the legal boundaries of a company, including lending, selling or making available for free download.

The typical license obligations cover:

- Inclusion of copyright and license in the source code or product.
- Documentation or information found in the user interface, so that downstream users know the origin of the software and what their rights are.
- Disclaimers of warranty on behalf of the authors.
- Notices as to source code availability.

With a vast amount of different Open Source licenses, the compliance process might seem overwhelming. However, in practice most organizations only use a limited number of licenses. Furthermore, there are only two major license categories: "copyleft" that requires companies to make the source code available; and "permissive" that applies minimal conditions, such as author attribution. For more information on licenses read the separate Section on Open Source and Copyright.

Why it is important

If the company breaches the license it may induce severe legal and business risks such as:

- **Copyright and patent lawsuits:** Copyright and patent damage compensations are dreadfully costly as intellectual property laws rule them. Intellectual property laws award a compensation for breaches based on perceived damages, whereas contract laws award a compensation based only on actual damages.
- **Injunctions, barring sales in a specific market:** This consequence is probably even more costly than damage compensations.
- **Loss of control of own software and Intellectual Property:** Legally you can't be forced to publish proprietary code under a copyleft license. However, considering the high costs of possible recalls, redesigns and compensations for copyright violation damages, it might be that the only viable option in the end is to publish proprietary code anyway.
- **A bad reputation that substantially hampers the business:** Regaining the trust and support of Open Source communities as well as the public will literally take years. Potentially, this could be the costliest of all consequences as you may lose decade long business opportunities.

The risks associated with not complying with Open Source licenses are often the initial driver for companies to get their Open Source practices in order.

Considerations

Since the very start of the Open Source movement licensing and copyrights have been a core ingredient. As there are business and legal risks attached to using Open Source, most companies recognize the importance of being compliant. This has led to that there is much information, good practices and different tooling options readily available. The main challenge is to implement a process that ensures compliance in an effective way.

Having a body that defines and governs the compliance process where all necessary disciplines (legal, development and management) are represented is key. This is one of the main areas covered by the Open Source Board (see pattern Org-4). Included in their responsibility related to compliance is to define what licenses can and cannot be used and decide on the inclusion of new license.

The end-to-end compliance process contains a set of activities to be performed to approve any new or updated Open Source component to be included in a product. To make it effective it is vital that compliance is checked already from the intake and maintained over the development cycle. Furthermore, it is recommended that most of the activities should be driven as close as possible to the development as it is a natural part of the software architectural design work and the considerations that come with them.

Since responsibility will be distributed in the company it is important to define a step-by-step process with clear roles and responsibilities. Subsequently much attention must be paid to adoption including information, training and coaching of development staff.

In addition to approval of each individual component, an inventory list of all Open Source components and their respective licenses must be maintained. See pattern Prod-2 Code Management for further details.

Further reading

- Haddad, Ibrahim (2016)." Open Source Compliance in the Enterprise". The Linux Foundation
- Haddad, Ibrahim, "Free and Open Source Software Compliance The Basics You Must Know" , http://www.ibrahimatlinux.com/uploads/6/3/9/7/6397792/0.pdf

Related patterns

- Org-1 Cooperation with Legal & IPR
- Org-2 Policies, Roles and Authorities
- Org-4 Open Source Board
- Proc-2 Control Intake
- Proc-5 Make-Buy-Share
- Proc-4 Control Contribution
- Prod-2 Code Management

Control intake

> Late identification of incompatibilities between proprietary and Open Source code or between different Open Source licenses can cause major rework or forced publication of proprietary code.

To ensure effective management of legal, business and technical aspects of Open Source software, it is important to control it already at the intake point. To this effect, a company should establish a decision process for intake of Open Source software that can be triggered both by a deliberate request to include Open Source and involuntary inclusions in code found by e.g. scanning tools.

What it covers

To manage intake of Open Source software in a controlled way, a process must be established. The process can be initiated by:

- An Intake Application Form in a checklist format (or in a tool) that can be used if the intake is the result of a deliberate request to include Open Source software.
- Scanning for Open Source software that either deliberately or involuntarily may have entered the system in different types of code, for instance through:
  - Proprietary code, either when new components are added to the system or as part of changes made to existing code.
  - Developer downloads.
  - Code from 3rd party suppliers.
  - Software bundled with hardware.

The main roles involved in the intake approval process are typically the Intake Officer, the OS Officer and Legal and IPR. Information that is needed as a basis for a decision include:

- Description of the Open Source component and the community behind it.
- Intended usage and business case for the intake. This should also cover security aspects and a technical or architectural analysis of the Open Source software component in the system context as background for understanding license and IPR impact.
- IPR and License information, and possible conflicts.
- Responsible team.

This information should be supplied regardless of how the process was initiated.

To guide the organization and the intake decision makers, there needs to be a clear definition of which Open Source licenses are deemed to be OK and Not OK to use (see pattern Proc-1 Control Compliance)

Why it is important

Open Source license requirements are activated at distribution, but the cost of initiating compliance activities first when releasing is much larger compared to doing it already at intake and maintain it through the development life cycle. Late identification of incompatibilities between proprietary and Open Source code or between different Open Source licenses can cause

major rework or forced publication of proprietary code (see pattern Proc-4 Control Compliance).

Another positive aspect of using a structured intake process is that it raises the general awareness of both legal (compliance and IPR), business (through a Make-Buy-Share analysis) and technical aspects of using Open Source software.

Considerations

To avoid waste, the company should keep a list of common licenses and how they are regarded with respect to intake:

- Blacklisted licenses – that will not be approved.
- Whitelisted licenses – that are generally OK to use, given the right context (like given that there is no risk for incompatible licenses).
- Pre-approved Open Source systems – that supply basic functionality and have been approved for usage across the product (like Google Android in mobile devices).

The process should deal with full features or functions rather than code snippets. This is not to say that identification of code segments that are copied from Open Source software do not need to be dealt with, but instead of pushing them through the process, they should be directed back to the responsible team to be re-written or to be considered in the context of a complete intake.

It is best to start out by applying the intake process in a strict way so that everyone involved can get familiar with the Open Source concepts and what is needed to take informed decisions. Decision mandate can be distributed as the Open Source maturity level is increasing in the company, thus speeding up the process.

By applying a Make-Buy-Share analysis (see pattern Proc-5) to all components in its software system, a company will supply general direction to the developers when it comes to where and how Open Source should be used and intake will require less analysis and be more standardized. Similarly, Code Management (see pattern Prod-2) will make the intake analysis simpler by supplying information on Open Source and licenses in other parts of the system. Remember that after intake, the component inventory needs to be updated.

Further reading

- Ibrahim Haddad, The Linux Foundation (2018), https://www.linuxfoundation.org/using-open-source-code/ [2018-03-20]

Related patterns

- Org-2 Policies, Roles and Authorities
- Org-4 Open Source Board
- Proc-1 Control Compliance
- Proc-5 Make-Buy-Share
- Prod-2 Code Management
- Proc-4 Control Compliance

Code review

> For any organization being serious about Open Source, peer review is one of the general practices that needs to be in place.

Code review is a well-known practice in the software industry, but often not consistently applied. However, having code reviews as an institutionalized practice is instrumental to be able to grow capability for any company being serious about Open Source software.

What it covers

The Code Review pattern is about making code review a common practice in the parts of the company that will have Open Source as part of their software strategy.

Code review is about making systematic examination of computer source code. It is intended to find mistakes overlooked in software development, improving the overall quality of software. Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections.

Code reviews have been a well-known practice in software industry for decades, and much information can be found on the subject, however code review is often not consistently applied to ensure the intended benefits.

Why it is important

Code review is recommended for all software development contexts as it enables early defect detection. It has e.g. proven to have significant impact on evolvability, thus very well suited for software with long life cycle.

For any company being serious about Open Source, peer review is one of the general practices that needs to be in place to grow Open Source capability and maturity.

Initially it is important when Open Source software is included in the code base, i.e. in the intake process where the code review will serve many purposes:

- Ensuring that the code does what is expected and is fit for purpose.
- Validating that the code fits in the software architecture.
- Detecting errors, issues and vulnerabilities in the code.
- Analyzing the coding standards and rules applied in order to decide on if and how to embed the Open Source software.

When starting to contribute software to an Open Source community, code review is equally important. The objective for any contributor is to get the code accepted as part of the next release. In order to achieve this, the functionality naturally needs to add substantial value to the code base. However, for the code to be accepted and for the company to be perceived as a serious contributor, the code needs to be of high quality and adhere to community practices. This will be very hard to achieve without serious code reviews.

If your company has the ambition to further influence and control the Open Source ecosystem, additional responsibilities will be added. This will in one way or another include ensuring the quality of the Open Source software releases covering also review and evaluation of other parties' contributions. Applying code reviews is instrumental in fulfilling those objectives.

Considerations

Since code review is a common practice, there is much information available on how to go about it. One example is included in the Further Reading list, "A Guide to Code Inspections". If you have not done systematic code reviews before, use available reference literature. as a starting point.

Make sure to decide on ambition, objective and method for your code reviews. It is more important to get started with less ambition, be persistent and learn as you go rather than spending much initial time in defining all details.

A general recommendation is not to include too much code in one individual review session. Focus is likely to get lost and quality may suffer. A few hundred lines of code per session is often seen as a suitable ambition level.

It is also recommended to investigate the potential in automating some of the code review tasks. Automation has become a more widespread and has often proven to save money as well as time.

Further reading

- Kolawa, Adam; Huizinga, Dorota (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. p. 260. ISBN 0-470-04212-5 Ganssle, Jack (February 2010). "A Guide to Code Inspections"(PDF). The Ganssle Group. Retrieved 2010-10-05.
- VDC Research (2012-02-01). "Automated Defect Prevention for Embedded Software Quality". VDC Research. Retrieved 2012-04-10.
- K. Wiegers "Peer Reviews in Software: A Practical Guide"

Related patterns

- Proc-2 Control Intake
- Proc-4 Control Contribution
- Proc-7 Create and Direct Communities
- Org-5 Open Source Community Culture

Control contribution

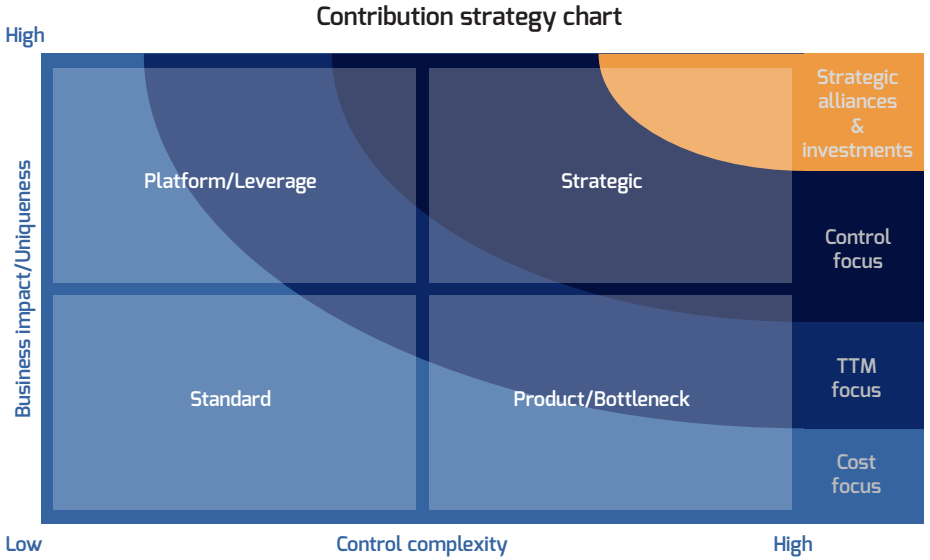
> It must be clear to all developers within an organization what the strategy for their Open Source components is and how the approval process for contributions works.

A company that wishes to increase the benefits of the Open Source software it is using, needs to do contributions and control the content of these contributions. This pattern describes how to manage strategies for, and approval of, contributions so they can be done in a controlled and efficient way.

What it covers

To make it clear for all stakeholders in your development organization that deal with Open Source code and that want to make a contribution to a community, a process needs to be established for how to get it approved. This process should entail:

- Classification of contributions in e.g. Trivial, Medium and Major contribution:
  - A Trivial contribution could for instance be a bugfix, and approval mandate for such contributions should be distributed.
  - Medium level contributions where expert consultation is recommended could for instance go through the Open Source Board.
  - Major contributions could be complete modules or frameworks involving complicated IPR deliberations requiring executive management decision.
- Roles – defining what Open Source roles and forums in the company are involved in the decision making around contributions (like legal and IPR, and the OS Board).



In addition to the approval process, a company should also formulate a contribution strategy for the different Open Source components it is using. This will be a further guidance for developers so that they don't spend unnecessary time and effort on contributions that are not in alignment with the strategic ambitions and are thus not likely to be approved. By looking at the Open Source components in terms of factors like their business impact and technical complexity (e.g. through the CAP model that is presented in the Further Reading material), such a strategy can range from 1) stating that only intake and bugfixes are relevant, to 2) having an active contribution strategy for a

component, and finally 3) deciding to take a leading role in a community. The introduction of a contribution strategy should also be reflected in the policy on contributions, see Pattern Org-2 Policies, Roles and Authorities.

Why it is important

It is through contributions that a company can gain influence in Open Source communities. Thus, it must be clear to all developers within a company what the contribution strategy for their Open Source components is and how the approval process for contributions works. It's fair to say that control of contributions is a central capability that needs to be established for a company that has taken a strategic decision to work with Open Source.

Considerations

When setting a contribution process, it is important to secure that it is swift, i.e. contains no unnecessary steps and has as short lead time for decision as possible. If this cannot be assured, you can be certain that your competitors will do contributions instead of you, your developers will lose interest and engagement and an opportunity to influence the Open Source communities is lost.

The Open Source contribution strategy defines the level of engagement and the process defines what approval steps to take depending on type of contribution (Trivial, Medium, Major). Generally, you should keep the decision authority (and handling of the different steps) in the process as hierarchically low and close to the developers as possible.

The contribution strategy should evolve along with the Open Source maturity in the company. Thus, the company is likely to initially only have a strategy to contribute bug fixes, moving to functionality contributions in areas where there could be cost reduction and shared development and finally identifying central components with a much more far-reaching contribution strategy when the company is ready for it.

The pattern Org-6 Grow Industry Experts includes a few suggestions on how to acknowledge and improve the level of contributions.

Further reading

- Linåker, Johan & Munir, Hussan & Wnuk, Krzysztof & Mols, C.E. (2017). Motivating the Contributions: An Open Innovation Perspective on What to Share as Open Source Software. Journal of Systems and Software. 135. 10.1016/j.jss.2017.09.032.
- https://opensource.guide/how-to-contribute/
- https://mashable.com/2011/03/30/business-open-source-communities/#PaoXXEVLwuqj

Related patterns

- Org-2 Policies, Roles and Authorities
- Proc-5 Make-Buy-Share
- Org-8 Collaborative Product Strategy
- Proc-7 Create and Direct Communities
- Prod-2 Code Management

Make-Buy-Share

> To become more cost effective by ensuring that you can focus properly on the differentiating components while sourcing off-the-shelf solutions in areas of less strategic value and use Open Source software for commoditized parts of the system.

A software product consists of many components. By doing a Make-Buy-Share analysis, a company can establish a holistic and strategic view of which components are suitable for own development, 3rd party solutions, out-sourcing and Open Source software.

What it covers

All companies with complex software-intensive products need to evaluate the different components in the system. Per component this means assessing:

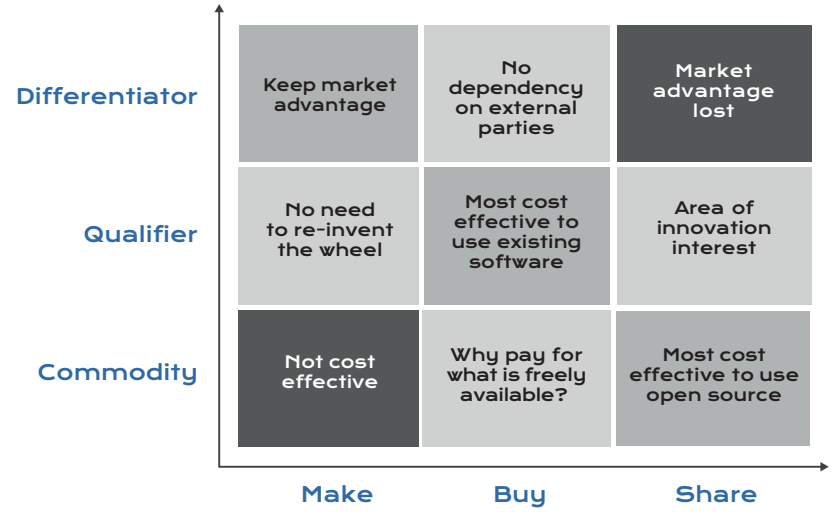
Is it:

- A commodity (thus not adding to the attractiveness of the product)?
- A qualifier (component adds specific value to the customer, but technology is not unique)?
- A differentiator ("only we can do it")?

Should we:

- Make the component ourselves?
- Buy it?
- Share it (use Open Source)?

The figure shows the Make-Buy-Share matrix that may be helpful as a way to understand what strategy is relevant for each of the cases. The Make-Buy-Share evaluation should not only be done once, but needs to be updated as the system evolves over time.



As guidance to the Make-Buy-Share decision (especially for understanding if a component is differentiating, a qualifier or commodity), the following parameters can be considered:

- **Availability** (of a certain feature or functionality): What 3rd party and Open Source solutions exist?
- **Cost:** What is the cost of development and maintenance for a component?
- **Criticality:** How strategic is the component for the product business in the longer term?

- **Competence:** Can current and future staffing needs be covered?
- **Resource load:** (Somewhat connected to cost and competence.) Does resource load for a specific component often reach critical levels thus risking delivery and quality?
- **Complexity:** What is the level of component complexity in terms of dependencies within the system and how difficult the technology is to acquire, develop, and control.
- **Marketing:** How marketable are the product features or functions?

Why it is important

One of the main reasons to apply the Make-Buy-Share strategy to your software system is to become more cost effective by ensuring that you can focus properly on the differentiating components while sourcing off-the-shelf solutions in areas of less strategic value and use Open Source software for commoditized parts of the system. With this strategy, you will also be able to increase innovation across the product through better-focused own development and joint development of qualifier and commodity components.

In addition, Make-Buy-Share decisions will guide the Open Source intake (pattern Proc-2) process by mapping the components in the system where Open Source software is wanted or unwanted.

Considerations

For the Make-Buy-Share analysis to become relevant, it is important to involve people like architects and product managers from in the company that have the right holistic mind-set and can judge the different parameters in a reasonably objective way.

A Make-Buy-Share mapping constitutes a static view of the system and needs to be continuously updated. To secure that this happens, it can be made part of the system governance structure. Looking at the figure, over time, components tend to move down along the y-axis and to the left along the x-axis.

Since the software industry is moving very fast, the input from the component owners to the Make-Buy-Share decision process is vital. Availability through 3rd party solutions or Open Source software will increase, and understanding of communities and crowd patterns (see pattern Proc-9 Crowd Based Requirements) may provide valuable insights. The Collaborative Product Strategy way of working (pattern Org-8) strengthens this communication within a company.

When looking at the parameters that affect the decision to Make-Buy-Share, especially the discussion around Criticality may become very subjective when involved stakeholders are trying to establish if a certain component adds customer or business value. In this case, the use of qualitative data (e.g. surveys) or quantitative data (e.g. usage statistics) can add further substance.

Further peading

- K. Petersen, D. Badampudi, S. M.A. Shah, K. Wnuk, T. Gorschek, E. Papatheocharous, J.Axelsson, S. Sentilles, I. Crnkovic, A. Cicchetti, “Choosing component origins for software intensive systems: In-house cots oss outsourcing or services? a case survey” Transactions on Soft. Eng.-in print 2017

Related patterns

- Org-2 Policies, Roles and Authorities
- Prod-1 Modularization and Control APIs
- Proc-2 Control Intake
- Proc-4 Control Contribution
- Org-8 Collaborative Product Strategy
- Proc-7 Create and Direct Communities
- Proc-9 Crowd-based Requirements

Frequent releases

> In essence, question all steps in the process and see what steps can be automated.

Having frequent software releases is strongly associated with concepts such as Continuous Delivery, Agile and DevOps, and many Open Source software projects have adopted a frequent release strategy. In order to ensure return on the investments made in Open Source, a frequent release strategy aligned with the release cycle of the Open Source software should be adopted.

What it covers

The Frequent Releases pattern is about releasing software to users on a frequent basis. It should be a common practice at least in the parts of the company that have Open Source as part of their software strategy. Separate parts of the company may have different types of solutions, where the Open Source potential varies. Also, if a company has one product, it may be possible to have independent release strategies for different parts of the product. This will however put further requirements on structure e.g. through decoupling of the software components. Read more in patterns Prod-1 Modularization and Control APIs and Prod-2 Code Management.

The Frequent releases pattern covers ensuring an adequate technical environment, implementing streamlined practices and providing the necessary tooling. One important step is to ensure that the production environment is set up and managed in a way that removes limitations in the release process. It often makes sense to assess virtualization of the production environment, e.g. having the environment in the cloud, as a part of preparing your infrastructure for frequent releases. Most important, however, is to apply good practices to ensure a stable, consistent and available environment. This covers e.g. version control, virtualized test of infrastructure and continuous delivery.

Furthermore, the actual release process must be effective including clear and delegated authority to decide on releasing software. This is more easily achieved if development is organized around small teams delivering more agile releases. To become successful the above must be supported by as much automation as possible of especially testing and the actual release itself. In essence, question all steps in the process and see what steps can be automated.

Why it is important

Many software organizations are still working with large releases with many bundled changes, however many have also started to apply frequent releases as part of their overall software strategy.

Their main reasons relate to increased speed and business flexibility through:

- Immediate market, user or community developer feedback.
- Early feedback on quality and stability.
- Less cumbersome release process.

The common adoption of frequent releases in Open Source projects usually involve users all over the world who eagerly download each new version as soon as it is released and test it as thoroughly as they can. For this reason, it is easily understood that any company having the ambition to play a key role in Open Source adoption, will need to implement frequent releases as a key feature in their software.

However, already when making the first inclusions of Open Source software in deliveries, there will be additional benefits in working with frequent releases. It is likely that new versions of Open Source software will be made available on a regular basis. If the company wants to get full benefit from the investments made in Open Source, it should have a release cycle that is aligned with the applicable Open Source projects, otherwise opportunities of having the best software in the market will get lost.

Taking the step to also become a contributor of software to Open Source communities will increase the importance of having the ability to release software on a frequent basis. The key reason to contribute software is to get the contributed version accepted by the community and thus becoming a standard. If you are not able to contribute your software swiftly to the community most likely someone else will get ahead of you.

Considerations

The release strategy is not something that can be derived separately, but must be put in context of the overall software process, software strategy and corporate culture. Since the long-term impact of these changes may be profound, it is recommended to run an improvement project with the necessary change management and adoption processes. The company's release strategy should also be aligned with Open Source communities' release strategies. This must be done with care and special attention must be paid to factors such as what are the Open Source communities of highest importance.

This is an area with a lot of terminology. Agile, DevOps, frequent releases, continuous integrations and flow are examples of related words and concepts. They will all mean different things to different people and different organizations. To avoid confusion, make clear to yourself what you want to do and what you mean.

Many companies have already started on this journey. Furthermore, many practices are defined and are readily available and the tooling for automation is catching up. In short, if you are serious about implementing frequent releases, there are proven approaches for you to use and become successful.

Further reading

- Antonio Cesar Brandão Gomes da Silva, Glauco de Figueiredo Carneiro, Fernando Brito e Abreu, and Miguel Pessoa Monteiro (2017); Frequent Releases in Open Source Software: A Systematic Review; MDPI, Basel, Switzerland.
- A. Deshpande, D. Riehle, "Continuous Integration in Open Source Software Development." Open Source Development, Communities and Quality. OSS 2008. IFIP – The International Federation for Information Processing, vol 275. Springer, Boston, MA

Related patterns

- Prod-1 Modularization and Control APIs
- Prod-2 Code Management
- Proc-4 Control Contribution
- Org-5 Open Source Community Culture
- Proc-7 Create and Direct Communities

Create and direct communities

> One of the main benefits is the sheer manpower and usage of a solution that can be mustered in a community. This will accelerate innovation, decrease time to market and improve quality.

Many software-intensive companies own software assets that are of general interest and high potential but do not contribute to the product differentiation. These assets are candidates for creating Open Source communities around, with the objective to harvest the advantages of community involvement e.g. increased innovation, decreased cost and time to market.

What it covers

There are some preparations to do before a community can be created. First and foremost, the company needs to identify good candidates to create an Open Source community around. The stakeholders must also find relevant responses to questions like: 1) what are the reasons for doing it? 2) how much (of the code) should be included?, 3) is there executive buy-in and budget for driving a project in terms of time, resources, costs, infrastructure etc.? and 4) is the project interesting for others and what participation can be expected from the start? All of this should be summarized in a project mission and a project plan.

After an internal agreement has been reached on creating a community, there needs to be a legal review to consider things like the impact on IPR, selecting an Open Source license for the code to be released, documenting license requirements, considering trademarks and deciding if contributor agreements are needed or wanted.

Along with the legal review, there are also several technical activities to execute before launch, for instance:

- Defining the governance structure and the basic processes for the community.
- Cleaning the code from dependencies and ensuring a consistent code style.
- Adding license and copyright information in the code library and files.
- Setting up the infrastructure including code repository, test environment and issue reporting. The infrastructure (like the tool chain) also needs to be checked for possible proprietary elements that may hinder participation.
- Creating communication channels like forums, wikis and social media channels.
- Provide relevant documentation, like community guidelines and usage examples.

When the roles that are relevant for governance have been assigned (e.g. a community manager or maintainer), the community is ready to established and the first development activities and contributions may take place. To get going as quickly as possible, use established best practices from similar communities, e.g. a kick-off event.

When the community is up and running and others have started to join, it is important to consider the obligations that follow with being a driver:

- **Communication:** Make sure that all changes going forward are communicated properly. If decisions are made in closed groups or meetings, these should be posted publicly so everyone in the community has access to the same information.
- **Taking control:** Some contributions will be out of scope, create unnecessary work for others or simply not be up to standard. These need to be gated in a polite and respectful way.
- **Mentoring:** Helping and guiding newcomers through the different processes in the community, not least the contribution flow.

As the community develops over time there will be new challenges. Some partners may want to drive the code in a different direction, which opens the question of the overall mission. Allowing forking or offering APIs and customization possibilities may be alternatives. To maintain a continuous healthy discussion in the community, it is a good idea to encourage and facilitate possibilities to meet face-to-face.

Why it is important

There are several reasons for why starting and maintaining an Open Source community can be the right thing to do. One of the main benefits is the sheer manpower and usage of a solution that can be mustered in a community. This will accelerate innovation, decrease time to market and improve quality. If the Open Source project draws enough external attention, it will also help to share (and if wanted: reduce) development cost.

Another aspect of creating a community is the engagement it opens up for – with people sharing your mission, with partners or even with customers. This will, for instance, give opportunities for your company to identify potential new employees and for your customers to do self-support through adapting or correcting code, thus shortening lead-time. Clearly a win-win situation.

Considerations

Starting and maintaining an Open Source community is no simple task, so therefore it is healthy to do a proper evaluation by asking yourself:

- Will your company actually manage to drive the Open Source project? Assess cost and capabilities.
- Can the same objectives be reached by joining an already existing project? Look around for existing alternatives.
- What is the probability that others will really join the project? Be honest with yourself.

Although participation in an existing community should be the first choice since it gives most of the wanted advantages (like speed, innovation power, quality), it doesn't necessarily offer the possibility to drive the direction of the project.

It is important to hold community culture in high regard. Things like "Release early, release often" suggest that:

- Shipping trumps perfection – to get feedback on new ideas or problems when there is still room for flexibility.
- Small increments rather than big bang releases – to maintain speed, visibility and improve debugging possibilities.
- Show by example and enforce code of conduct to maintain a good community culture.

Finally, make sure that focus is kept – i.e. keep down the number of active discussion threads you are driving to avoid fragmenting the community mission. And remember not to take yourself to seriously.

Further reading

- G. von Krogh, S. Spaeth, K.R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study", Research Policy, Volume 32, Issue 7, 2003, Pages 1217-1241
- Nicolas Ducheaut, "Socialization in an Open Source Software Community: A Socio-Technical Analysis"

Related patterns

- Org-7 Developer Program
- Org-5 Open Source Community Culture
- Org-6 Grow Industry Experts
- Org-8 Collaborative Product Strategy
- Proc-10 Create and Govern Ecosystems
- Prod-3 Creating a Software Platform

Industry-wide collaborations

> When you are a big player in a given business, you have strong influencing power to establish your solution as an industry wide standard. When you are a smaller player, you can join with other small players to challenge the bigger players.

Companies working with Open Source establish industry-wide collaborations to grow their business, increase revenue and contribute to creating standards. The collaborations can be centered around a software platform or an ecosystem.

What it covers

The industry wide collaborations pattern is about actively seeking collaboration partners that can help to grow your business, expand your product offering or reach new markets with your Open Source software platform or ecosystem.

From an engineering perspective, one starting point for identifying potential collaborations is through the contributions that are made to an Open Source project that the company is engaged in. Accepted contributions could trigger discussions between developers about possible collaborations. Another strategy is to publish a list of open problems or challenges associated with a given Open Source platform or technology online, and check if responses can point out collaboration partners.

However, since the objective is mainly business oriented, managers and leaders in the company need to take the lead in reaching out and establishing collaborations. This can be done through the contacts and relations they have created thanks to engagement in Open Source platforms or ecosystems. Management involvement is important since developers often have little or no decision authority in establishing business-based collaborations or discussing details about value creation and revenue sharing. The Open Source context also lowers barriers between companies since is diffuses potential IPR conflicts.

Why it is important

The main benefits for a company to establish industry-wide collaborations include:

1. Usage of the Open Source platform or ecosystem is expanded outside the company's boundaries and current markets, with a potential to grow business and increase sales.
2. They provide a way to find complementing knowledge or experience and to realize new business ideas with the help of partnerships.
3. They create a win-win situation where collaborating partners learn from each other and get to understand each other's business environment, opportunities and constraints. Big players with a strong influencing power can establish their solutions as industry-wide standards. Smaller players can join with other small players to challenge the big players together.
4. Collaborations are one of the driving engines for transforming an Open Source ecosystem into an industry-wide standard.

Considerations

The main consideration when starting a collaboration is to understand who benefits from it and how to co-create value and share revenue. A company will need to carefully select its partners since these decisions usually have long-term implications.

Although there may be a healthy reluctance towards sharing too much with others, there is also the fact that high entry barriers will eliminate many potential collaborations. Thus, there is a need to find the right balance between openness and exclusiveness.

When setting up collaboration around an Open Source project, communication channels and routines for knowledge exchange and idea discussions need to be established. Some examples include notifications about new contributions and their potential in an Open Source ecosystem context, organizing hackathons for developers interested in a given technology or organizing development days.

Finally, regulatory activities may be needed when you establish industry-wide platforms and solutions. This happens when entering domains where compliance and certification towards standards and market regulations are required.

Further reading

- M. A. Storey, A. Zagalsky, F. F. Filho, L. Singer and D. M. German, "How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development," in IEEE Transactions on Software Engineering, vol. 43, no. 2, pp. 185-204, Feb. 1 2017. doi: 10.1109/TSE.2016.2584053
- Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. "Two case studies of open source software development: Apache and Mozilla". ACM Trans. Softw. Eng. Methodol. 11, 3 (July 2002), 309-346.
- https://www.openautoalliance.net/#about

Related patterns

- Proc-4 Control Contribution
- Proc-7 Create and Direct Communities
- Proc-10 Create and Govern Ecosystems
- Prod-5 Open Source Driven Platform Innovation

Crowd-based requirements

> Utilizing Crowd-Based requirements engineering means mobilizing as many crowd members as possible to communicate and discuss their needs regarding the evolution of existing software products.

Crowd-based Requirements Engineering enables harvesting external sources of requirements and ideas discussed in Open Source communities. It also helps to extract requirements from the product usage data. It also enables more time efficient requirements gathering as a large of requirements discovery is automated and done by the crowd. This pattern is on level 4 as it improves requirements engineering efficiency and effectiveness based on the benefits from Open Source community creation, governance or participation. It expands the make-but-share analysis by understanding the requirements and future needs for commodity, qualifier and differentiator components.

What it covers

The Crowd-based Requirements Engineering pattern is about creating mechanisms to harvest external opinions from the crowd – a large set of users and Open Source community member who continuously provide feedback. Crowd-Based RE is based on the crowdsourcing principles in which individual or organizations use contributions from Open Source communities to obtain ideas about future needs for the products.

Companies working with Open Source ecosystems (both contributing and governing them) are exposed to large amount of information (business intelligence, product usage data, reviews and other forms of feedback) and data. The data is heterogeneous, multi-sourced and challenges requirements capturing and analysis activities. Utilizing Crowd-based Requirements Engineering means mobilizing as many crowd members as possible to communicate and discuss their needs regarding the evolution of existing software products.

Requirements Engineering for Open Source is based on informalisms for describing Open Source requirements and what developers are currently working on in the Open Source projects. Informalisms are informal, online documents that often are created after the implementation is ready and often capture the detailed rationale, contextualized discourse, and debates for why changes were made in development activities, artifacts, or source code files.

The main benefits for organization to use Crowd-based Requirements Engineering include:

1. Mobilizing many crowd members to provide continuous feedback and suggestions regarding potential product requirements and main issues (complement market predictions).
2. Minimizing the cost of obtaining feedback and recruiting potential users and enabling software application usage and context monitoring (increased efficiency of requirements engineering).
3. Reducing the feedback time between introducing a new requirement to receiving customer feedback about it and receiving just-in-time feedback from multiple channels anytime.

4. Scaling up requirements elicitation in a cost-efficient manner and ensuring that multiple-opinions are considered. Supporting requirements triage and decision making with continuous opinions and sources of requirements.
5. Automating or semi-automating work-intensive parts of requirements elicitation, triage and pre-screening and giving more time to analyze the results of this screening and make decisions that will benefit products.
6. Creates a platform for requirements engineering process automation and data-driven requirements discovery.

Why it is important

In traditional software engineering, business analysts identify user or market needs (intelligence), synthesize a set of features and functions that satisfy those needs as requirements specification (design), and prioritize and package these requirements based on business strategies and constraints (choice). This process is often not effective and is not sufficiently scalable as it funnels a small amount of information on the users' needs through a limited capacity that struggles to associate and recombine that input to provide better and more innovative products.

Crowd-based Requirements Engineering utilizes the potential that the crowd brings, and the informal feedback provided by the crowd to find valuable product requirements. A motivated crowd provides continuous feedback that improves requirements validation and just-in-time feedback loop and allows for large-scale experimentation (early adopters, technology experts). The crowd helps to reduce requirements validation time and cost.

Considerations

Automating crowd data analysis is critical here and includes defining: 1) what data sources are relevant, 2) how to filter this data and injects to company's requirements processes and 3) how to synchronize with internal product strategies.

Further reading

- https://en.wikipedia.org/wiki/Crowdsourcing.
- Scacchi W. (2009) Understanding Requirements for Open Source Software. In: Lyytinen K., Loucopoulos P., Mylopoulos J., Robinson B. (eds) Design Requirements Engineering: A Ten-Year Perspective. Lecture Notes in Business Information Processing, vol 14. Springer, Berlin, Heidelberg.
- E. C. Groen et al., "The Crowd in Requirements Engineering: The Landscape and Challenges," in IEEE Software, vol. 34, no. 2, pp. 44-52, Mar.-Apr. 2017.

Related patterns

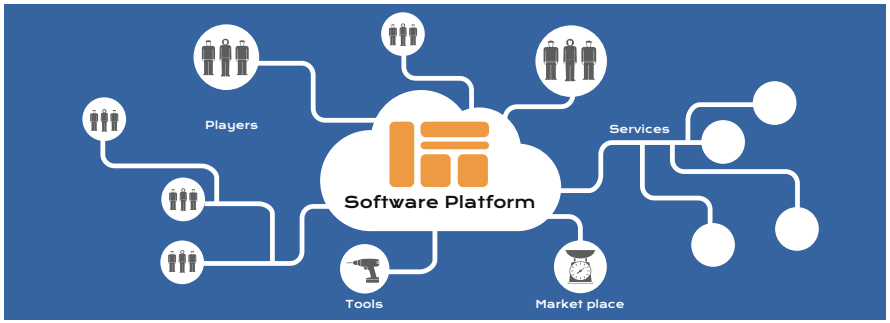
- TBD

Create and govern ecosystems

> Open Source ecosystem give several benefits from Open Source involvement, including value co-creation, contributions and sharing development resources and creating additional value streams for the platform or marketplace owner.

Creating and governing ecosystems is about creating communities of communities based on Open Source involvement. An ecosystem requires an Open Source based platform and a marketplaces where additional features and services can be added. A successful ecosystem can become an industry-wide standard that will accelerate product innovation, expand market penetration and enable additional revenue streams.

What it covers



The main elements of the Open Source ecosystem are presented in the figure. An Open Source ecosystem is a community of communities that has the following key elements:

1. **An Open Source platform** – a common Open Source code base that has an associated community. (Also see pattern Prod-3 Creating a Software Platform)
2. **Players** - the Open Source platform is driven or used by different types of ecosystem players (like platform leaders, niche players and bridge players), vendors, customers and external users and actors. The platform leaders are typical orchestrators (responsible for governance) that largely determine the growth of an ecosystem.
3. **An established governance model** – with an explicit governance structure and processes, ecosystem orchestrators and communication and contribution channels. Platform leaders use the governance model to grow the ecosystem and ensuring that it reaches its strategic goals.
4. **A market place** – many ecosystems are centralized around a market of extensions (like app stores or app markets). This provides the ecosystem with a well-defined market, which attracts strong players. A defined market place makes it simpler to identify the key actors and the relationships between them. The market place can also supply additional hardware and software services to the platform.
5. **Lock-in mechanisms** – a mechanism that prevents the ecosystem players from switching to competing Open Source platforms too easily. Switching should be possible but associated with significant technological or financial costs. This helps to ensure that prominent players stay with the platform and help to develop it.
6. **Data access mechanisms and policies** – this includes data probe access to source code and code interfaces and rules to accessing and sharing data generated by the executed code. Data access is important for creating market extensions (apps) that can work on data and reuse platform code.

7. **Revenue sharing mechanisms** – includes defining symmetric and asymmetric revenue sharing models for ecosystem leaders and other players. This can for instance be per-usage revenue models or advertising-based revenue.

Infrastructure and communication channels established during the community creation to support participation need to be maintained over time by the ecosystem governance. If required, ecosystem specific infrastructure and tools need to be expanded. The general state of the ecosystem can be visualized by health measures that should be introduced and regularly evaluated by the ecosystem governing company.

Transparency and a clear definition of the ecosystem and its governance structure should also be derived and maintained. Without a shared understanding of the ecosystem it is very difficult to have a clear and open ecosystem strategy. Additional governance aspects include coordination of contributions to other ecosystems, formalization of entry requirements for new participants and creating customer and partner directories.

Why it is important

A company that governs an Open Source based ecosystem has the opportunity to steer development effort towards its business agenda and can therefore partly focus internal development resources on creating (proprietary) differentiating parts.

By creating ecosystems, a company accelerates product innovation for the products extracted from the software platform. It also gives the company a competitive advantages by helping to establish a market position or increase the penetration of a given market, to a level where it can act as a powerful market disruption mechanism.

Open Source ecosystems enjoy the general benefits from Open Source involvement, including value co-creation, contributions and shared development cost and creating additional value streams for the platform or market place owner.

An ecosystem which adheres to the transparent and collaborative principles of Open Source is not only a powerful knowledge exchange platform, but the underlying software platform has a possibility of becoming an industry standard. This enables penetration of new markets and business areas.

Considerations

The biggest concern for a company that is taking the role as a governing organization, is to understand that Open Source ecosystem creation is a long term commitment. Internal industry experts need to be assigned roles in the ecosystem governance structure and this will be an ongoing effort for as long as the ecosystem exists. Without long-term management support, the ecosystem communities will not receive sufficient support to grow and mature.

Additional considerations include:

- Understand your business environment including:
  - How your current software or hardware services interact with other stakeholders.
  - How to create a platform and to understand if this platform can become interesting for external players.
- Having a stable platform with established interfaces and creating a community culture are mandatory prerequisites. Additionally, supporting Open Source collaboration mechanisms in the governing organization is mandatory for a successful Open Source ecosystem launch and growth.

Further reading

- Baars A., Jansen S. (2012) "A Framework for Software Ecosystem Governance". In: Cusumano M.A., Iyer B., Venkatraman N. (eds) Software Business. ICSOB 2012. Lecture Notes in Business Information Processing, vol 114. Springer, Berlin, Heidelberg
- Slinger Jansen, Sjaak Brinkkemper, Michael A. Cusumano. 2013. "Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry". Edward Elgar Publishing, Incorporated.
- Jansen, Slinger, and Michael Cusumano. "Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance." Proceedings of IWSECO (2012): 41.

Related patterns

- Org-6 Grow Industry Experts
- Proc-7 Create and Direct Communities
- Org-9 Self-managed Organization
- Prod-3 Creating a Software Platform
- Proc-8 Industry-wide Collaborations
- Org-10 Directed by Business Aspects
- Org-11 Authority in Open Source

Anti Patterns

Shun the “use, but not contribute” trap

> An active Open Source contribution strategy has considerably larger advantages than disadvantages should be shared through communication and training.

Many companies will get stuck in thinking that only using Open Source fulfils all their needs and there is little benefit compared to cost and effort in active participation in Open Source communities. They have still not realized that passive consumption of Open Source is a flaw and need to continue their transformation to better leverage on Open Source and shun the “Use, but not contribute” trap.

The trap

A common misconception among companies that have started to include Open Source software as part of their solution is that they stick to only using it. They are satisfied with just getting access to what is seen as cool and free software that solves a problem. Thus, they will refrain from doing contributions and more generally to interact with Open Source communities. Either this option is not considered at all or it is regarded as being complicated and costly. However, in doing so they will need to face up to facts like:

- They will have no insight in the direction of the Open Source communities and important new releases will always come as a surprise.
- They will not able to influence the communities with requirements from the outside (communities are driven by an internal “itch”).
- The large communities generally move fast (e.g. Linux: 9000 commits/day, Chrome release cycle: 6 weeks).

The result of not contributing is that the company will accumulate a lot of own patches to the Open Source software components in their system to fix issues or to tailor them to their specific product needs. Together with the above-mentioned facts this leads to:

- Patch management, which will slow down development when new versions of Open Source software need to be integrated:
  - The extra effort required due to patches may even discourage from keeping Open Source software components updated – at the risk of missing important bug and security fixes.
- Puts the full burden of maintaining patches over time on the company itself.
- The company cannot be part of community innovation and is not likely to be open to adjusting its roadmaps in accordance with the development direction in the communities. This constitutes missed opportunities.
- Over time, the collection of patches will become more of a liability than an asset. As the Open Source communities release new and better versions, the investments in patches could be made obsolete and worthless.

This is a trap that many companies end up in. Thus it is vital to turn it all around and look at the benefits that can be gained by opening up for contributions and the effects this will have on the development organization.

How to get out of the trap

The key to getting out of the trap lies in understanding the importance of contributions and participation in Open Source communities. There are three major reasons for contributing:

- It reduces cost of maintenance:
  - Once a patch has been contributed, it will be maintained by the community.
  - Your contributed patches force competition to re-adjust their set of patches, thus inflicting them a cost.
- It improves time to market:
  - With fewer patches to maintain, development can focus on differentiation and can more easily keep Open Source software components updated.
  - By participating actively in the Open Source communities that are vital, insights on where they are headed will be gained, allowing for adjustment of requirements. This reduces the risk for late and costly redesigns that can delay product launches.
- It opens the possibility to influence communities:
  - By letting your engineers climb the meritocracy ladder of Open Source communities, they will create relationships that can leverage your ideas and innovations.
  - Eventually this may lead to an influencing position that can help to drive your business agenda further.
  - This must, however, not be pursued to the extreme. A community will not give up control to members that only work for their own best interest.

The understanding that an active Open Source contribution strategy has considerably larger advantages than disadvantages should be shared through communication and training. There may, however, still be reluctance towards doing contributions since it is seen as giving away software that is perceived to have high value, and product owners may be concerned about losing control of the code. The way forward lies in defining what strategy to use for the different components in the system (as described in the pattern Proc-5 Make-Buy-Share) and to make use of the advantages achieved from active Open Source participation (as shown in the pattern Org-8 Collaborative Product Strategy). Also, patches in Open Source software components very rarely retain value over time.

Apart from supplying training and defining contribution strategies, the development organization will need clear guidance on how do contributions (as presented in the pattern Proc-4 Control Contribution). Thus it will finally be able to free itself from the trap of "using but not contributing"

Further reading

- J. Linåker et al. "Motivating the Contributions: An Open Innovation Perspective on What to Share as Open Source Software", Journal of Systems and Software, 2017

Related patterns

- Proc-5 Make-Buy-Share
- Proc-4 Control Contributions
- Org-8 Collaborative Product Strategy

Acknowledgements

Scaling Management Framework

The Industrial Open Source patterns builds on results created in the Scalare project (www.scalare.org), a pan-European effort between industry and academia in Sweden, Ireland, Germanya and Spain. The project was supported by Enterprise Ireland, Science Foundation Ireland and the Swedish innovation agency, Vinnova.

Orion project

The work is partially supported by a research grant for the ORION project (reference number 20140218) from The Knowledge Foundation in Sweden.

Industrial Open Source

A new wave of Open Source development being led by industrials and companies are growing with open source at the heart of their business. These companies are using Open Source to build commercial products. They are creating new business models that are allowing them to succeed in emerging business domains using technologies such as AI, Cloud and IoT. Consequently, today's industry faces a complex environment where systems are no longer primarily built on proprietary development, but on a mix of in-house, third party and open source.

This is a handbook on how to create and drive an Open Source Program in an industrial setting.