InnerSource - An Introductory Tutorial

What is this turorial for?

The commonly used definition of InnerSource states that it is *"the use of best practices from successful open source projects and establishment of an open source culture within your organization"*.

However, for the uninitiated, that may not convey a lot of meaning – what are these so-called *"best practices"* and what does *"open source culture"* even mean? If you have those questions, this tutorial is for you!

The InnerSource Commons community, the go-to community for all InnerSource practitioners around the globe, has an excellent set of resources but they assume some amount of prior knowledge about the mechanisms of how Open Source and InnerSource works and uses a number of ad-hoc terminologies. Our goal in this section and the next is to prepare you so that you can better understand what they are talking about and familiarize yourself further with InnerSource.

Is this tutorial for you?

Do you need to work through the tutorial? Take the quiz below to find out.

OSS Introduction

Open-source software (OSS) is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to use, study, change, and distribute the software to anyone and for any purpose as long as it adheres to the license restrictions (e.g. some OSS licenses explicitly forbid monetizing the product for profit and almost all licenses require the name of the original author(s) of the software be mentioned in any future distributions).

In contrast, the primary business model for closed-source software involves the use of constraints on what can be done with the software and the restriction of access to the original source code. The end result is that an end-user is not actually purchasing software, but purchasing the right to use the software. The source code to closed-source software is considered a trade secret by its manufacturers.

The top four reasons (as provided by Open Source Business Conference survey) individuals or organizations choose open-source software are:

1. Lower cost
2. Security
3. No vendor 'lock in'
4. Better quality

Both the number of OSS projects and the number of developers contributing to OSS projects have reached staggering heights, and OSS has become a critical component of our modern digital infrastructure, for example, you might be familiar with the widely popular **LAMP** stack, namely, **L**inux, **A**pache, **M**ySQL, **P**HP/**P**erl/**P**ython, which is a set of open source software widely used for web application development.

The topic of OSS is vast, so we will stop the introduction here. There are numerous resources about OSS that you can read to know more about it. We suggest you to start with the Wikipedia page for Open Source Software.

OSS Development Basics

Open Source Software development is typically very different from the software development process within a company. The open source model is a decentralized software development model that encourages open collaboration. Open collaboration is defined as: *"any system of innovation or production that relies on goal-oriented yet loosely coordinated participants who interact to create a product (or service) of economic value, which they make available to contributors and noncontributors alike."* and many of the characteristics of OSS development are derived from this development style.

The Cathedral and the Bazaar

In his 1997 essay The Cathedral and the Bazaar, Eric S. Raymond highlighted the differences between the traditional on-premise software development and the open source development.

Raymond likens the development of software by traditional methodologies to building a cathedral, *"carefully crafted by individual wizards or small bands of mages working in splendid isolation"*. Here development takes place in a centralized way. All roles are clearly defined. Roles include people dedicated to designing (the architects), people responsible for managing the project, and people responsible for implementation.

In contrast, OSS is often developed in a bazaar style, which Raymond described as *"a great babbling bazaar of differing agendas and approaches."* In this model, roles are not clearly defined. However, it is far from pure chaos that some people think it is - it is typically a bit messy but people often find what they are looking for. A few key characteristics of this bazaar style development are:

- **Users should be treated as co-developers:** The users are treated like co-developers and so they should have access to the source code of the software, and are encouraged to submit new features, code fixes, bug reports, documentation, etc. Having more co-developers increases the rate at which the software evolves. Linus's law states, "Given enough eyeballs all bugs are shallow." This means that if many users view the source code, they will eventually find all bugs and fix them.

- **Early releases**: The first version of the software should be released as early as possible so as to increase one's chances of finding co-developers early.

- **Frequent integration**: Code changes should be integrated (merged into a shared code base) as often as possible so as to avoid the overhead of fixing a large number of bugs at the end of the project life cycle. Some open-source projects have nightly builds where integration is done automatically on a daily basis.

- **Several versions**: There should be at least two versions of the software. There should be a buggier version with more features and a more stable version with fewer features. The buggy version (also called the development version) is for users who want the immediate use of the latest features, and are willing to accept the risk of using code that is not yet thoroughly tested. The users can then act as co-developers, reporting bugs and providing bug fixes.

- **High modularization**: The general structure of the software should be modular allowing for parallel development on independent components.

- **Dynamic decision-making structure**: There is a need for a decision-making structure, whether formal or informal, that makes strategic decisions depending on changing user requirements and other factors. Compare with extreme programming.

Product vs. Project

Another key difference between the traditional software and OSS is that OSS is often distributed without a guarantee. This is because in contrast to a typical software product, OSS software are often ongoing projects. The source code is made available, so the onus is on the user to make it work as they intend it to work. However,

typically users can and do get support from the community, where more experienced users might suggest ways to fix the problems.

OSS Communities

You must have noticed in the previous sections that the term “community” has appeared several times. In fact, the importance of an engaging community for the success of OSS can’t be overstated. The users who turn into contributors, help new users with any problems, and attract more users by spreading the word are absolutely essential for an open source software to flourish.

OSS developers are also acutely aware of this fact, in fact, the motto of the Apache Software Foundation, a community of a large number of hugely successful OSS projects, is *“Community over Code”*.

InnerSource Introduction

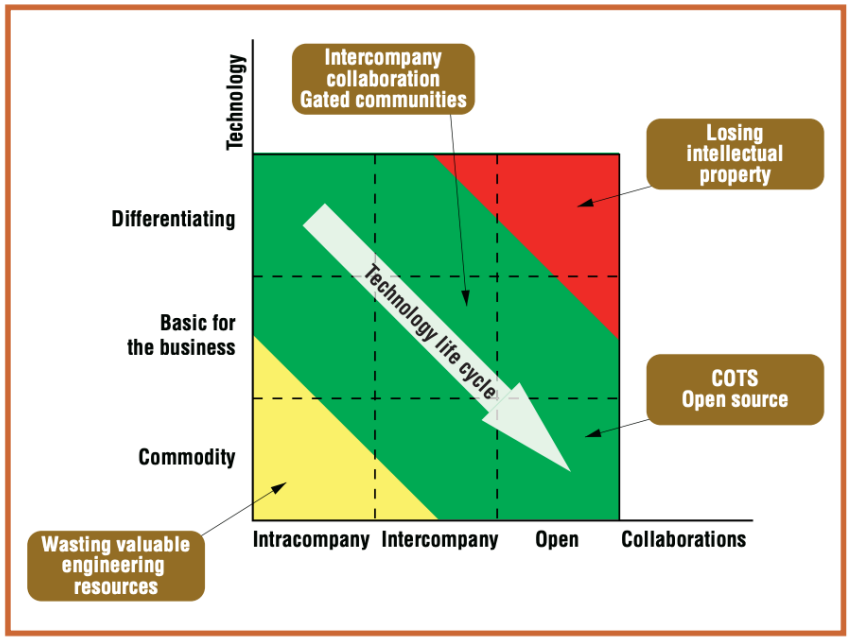
In this section, we will try to break down the jargons and terminologies used by the InnerSource community and try to lucidly explain the key ideas about InnerSource.

OSS Best Practices and Culture

Just as every project in a company are different, every OSS project is different. In fact, OSS projects often show a lot more diversity since people all over the globe can contribute to an OSS project. However, software engineering practitioners and researchers have identified a number of practices undertaken by many of the successful OSS projects that they believe are beneficial for more efficient software development in general. The term “OSS Best Practices” refers to the these set of practices, referring to how OSS projects handle the creation of requirements, selecting which features are to added, code creation, code review, bug fixes etc.

The “Open Source Culture” refers to how the contributors to a project treat each other and how the privileges are distributed. Typically, OSS follows a meritocratic system where the people who contribute the most to a project are given the most privileges and honors. The importance that is given to creating and fostering a community is also another hallmark of the OSS development. Many of these cultural values are different from the culture within a company, where the management structure tends to be quite different.

In is not always easy to integrate these practices and cultural values with the typical software development process within a company since the governance structure tends to be different and pretty rigid. However, many of the end goals are also different and, just like many other things in life, there is no “one-size-fits-all” solution to software development. Van der Linden et al., in their 2014 paper *“Commodification of Industrial Software: A Case for Open Source”*, showed where open style development (Open Source/InnerSource) is most suitable in the following plot:



Revisting the InnerSource Definition

Let us revisit the definition of InnerSource with the context described above. The goal of InnerSource is not to completely replace the existing software development practices, but to leverage the knowledge and benefits of Open Source development where it is most suitable. In contrast to OSS development, the resulting software is not openly distributed and, almost always, the contributors to the “InnerSource” projects are the developers within the same company. So this is like creating a microcosm of Open Source within the company.

Why InnerSource Matters

As you saw in the figure earlier, some types of software developed within a company can benefit a lot by adopting the Open Source style development. These benefits are amplified for larger companies. This fosters code reuse, so that different departments do not have to write and maintain separate code from scratch for doing very similar things. This also helps in better understanding of the requirements for the software which is used by different departments and, by virtue of having more users, improves the quality and fosters innovation.

Arguably, the biggest benefit of InnerSource, i.e. adopting the Open Source style development, is breaking down the silos and knowledge sharing. Teams that work in very niche projects for a long time tend to develop their own “hacks” for doing certain things that might be completely unknown to the rest of the people in the company. This works as long as status quo is maintained, but, if a new developer joins the team or if some important member of the team leaves, it becomes extremely problematic. A similar situation might happen if a team was using some tool from a third-party vendor, and the vendor leaves - the tool might become unusable. These risks can be mitigated by adopting InnerSource for such projects, which keeps the knowledge accessible to the relevant members of the company and is beneficial for the company in the long run.

InnerSource Terms Glossary

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]