

**developer.\*****The Independent Magazine for Software Professionals**

## Open Source Strategies for Software Developers

by Johanan Lancaon

### Introduction

Analysts tout 2005 as the year of open source. Its use has gained increased media attention, and many software consulting companies are starting to support various open source projects. As developers, we know there is a definite benefit gained with using open source software in some aspect of our project, whether it is used as a tool (e.g. Eclipse IDE, gcc, etc.) or integrated deeply into the project (e.g. Tomcat, Spring Framework, Apache HTTP, etc.).

In fact, most of us use this type of software without much thought to the issues surrounding its use. We hear about a great new open source software project, download the latest release, deploy the "Pet store" example, develop the "Hello World", and on we go using this software in every other project we work on. Open source software gives the developer more options, helps to increase the knowledge base among developers, and improves the overall programming prowess of the community.

With all the benefits associated with open source, developers might wonder what the issues are surrounding the use of open source software. What are the risks? Is this software really free? What if I need support? These, and several others discussed below, are all valid questions a developer should ask when considering whether an open source product should be incorporated into the developer's project.

### Open Source: A Definition

Let's take a step back and define what open source software means. Some people use the term loosely to describe any software that is freely distributed with modifiable source code. An organization called the Open Source Initiative (OSI) maintains a more formal and strict definition on its web site (<http://www.opensource.org/docs/definition.php>). The OSI uses ten criteria in its definition, addressing issues such as the ability to freely distribute software, accessibility to source code, ability to derive works, and more.

The OSI also reviews and approves open source licenses to determine whether they meet the organization's standards for open source licensing. At the time of this writing, there are 58 different licenses approved by the OSI, including the GNU General Public License (GPL), the Apache License 2.0, and the Mozilla Public License 1.1 (MPL). Not all "open source" licenses, however, meet or exceed the OSI's criteria.

## Communicating with Management

Typically, the developer finds management to be of two different mindsets when it comes to open source. They are either supportive because they believe it is an inexpensive solution, or they are not supportive because of the perceived risks associated with its use (this may stem from not understanding the paradigm, or not having the ability to purchase support).

In the first case, the developer may also find management to be overly enthusiastic in the use of open source software. This usually stems from the initial cost-savings aspect of using open source software. Often times, management hears how open source will ultimately save a project a lot of money. This may be the main reason why management will opt for the use of open source software. An example of this may be the migration of a software development project that has been implemented using a non-open source database such as Oracle 10g to an open source database (e.g. MySQL, PostgreSQL, or Firebird).

However, management may not understand the risks associated with migrating from one database to another. They may also be unaware of the process, time, and resources necessary to complete this project successfully. The developer may need to explain to management the risks and any potential roadblocks that could be encountered during the migration.

In the second case, the notion of open source software may seem enigmatic to management. The fact that source code is available may be thought of as a security risk for the company. Management may treat the lack of up-front cost for some open source projects as adding to overall risk. They may even think the product is not serious enough to be used in a mainstream production environment or software product.

In this scenario, the developer must take on the responsibility to educate management as to the capabilities of the open source product, and how the use of it can both help the "bottom-line" as well as improve the quality of the project.

As discussed in the next section, management should also know about the licensing ramifications associated with distributing any type of software. Often times, the developer is the only one who fully understands the license. In fact, many times the license is only available in the compressed source download. (Interestingly enough, it is a requirement to read the license before proceeding with uncompressing the file for some open source projects.)

Again the responsibility falls to the developer to properly educate management in any licensing issues that might occur. If management does not appear to understand the notion of open source, the developer may need to provide any necessary information as it pertains to open source prior to the explanation of the specific license. At times, the developer may be the sole educator in terms of open source understanding and licensure in the company.

## Licensing

I know what you are saying, "Who really reads those, anyways?" Well, I do (the lawyers told me to say that). Remember what your parents said? Eat your veggies, take your vitamins, button up before you go out in the cold, and make sure to read that license before using any open source software (I would make sure to read licenses of non-open source software too, even though those are generally long and frightening).

Let's say a developer is creating an application that will be sold commercially, and let's say the developer integrates a component released under the Apache license. Using examples from a few different open source licenses, typical questions include,

- What are some of the specifics we must take into account in distributing the Mozilla software product with my application?
- Can I use parts of an Apache product in my application but not the whole thing?
- Since I am using a GNU product (which is released under the GNU GPL license), do I have to release my software under the GPL License?

Let's take the first situation. Assume the product we are selling requires Mozilla Firefox to be included in the distribution. In addition to this, let's assume the Firefox source code was modified. The first thing to determine is which Mozilla license Firefox is distributed under. According to Mozilla.org, it's products are distributed under the MPL or the Mozilla End-User-License-Agreement (EULA). Looking at the license file on my Firefox installation, I noticed the MPL 1.1 is what Firefox is distributed with for Windows.

Next, it is necessary to determine what was done to the Firefox browser. Since the Firefox source code was modified, there are procedures that must be followed to properly re-distribute the software. For instance, it is necessary to make source code available for modifications that were performed on Firefox. In addition, it is necessary to 'label' the changes that were made to the Firefox source code. According to the Mozilla.org site, it is possible to do this by way of diffs. The FAQs posted on the Mozilla.org site contain more information regarding the specifics that need to be taken into account in this situation (<http://www.mozilla.org/MPL/mpl-faq.html>).

The second scenario involves using a component of an Apache product, but not the whole thing. Let's say we are interested in using parts of the Apache Jakarta Commons code base. Apache makes separate components available for download such as the commons-pool component. This is a set of JAVA packages used for object pooling. The Apache License 2.0 addresses "Derivative Works" which is defined as any work in Source or Object form that is based on the specific Apache software project code. Similar to the Mozilla license, it is necessary to annotate any changes that were made to the source code if we choose to only use parts of it, and then properly include the license file as well as a place for anybody to obtain the modified source.

Our third scenario involves GNU. This one gets a little tricky. If you choose to use code that is distributed by GNU and it is distributed under the General Public License (GPL), then your code must conform to the GNU license. In fact, the license you must use to distribute your product may have to be the same GNU license. This is where open interpretation can get a little messy.

Traditionally, software that uses a library that is released under the GPL is required to be distributed under the same GPL license if the developer chooses to distribute the product (note: The GPL allows the developer not to distribute the product). This is not to say that it is not possible to use GNU-licensed software and proprietary software in conjunction with each other. To do this, the software must be logically separate as mandated by GNU (For more information, please see <http://www.gnu.org>).

Since these licenses all conform to the notion of what the OSI terms "open source" the answers should be very similar. However, there are slight differences that must be taken into account.

What happens if the open source license does not conform to the OSI specification? This does not necessarily mean that you should not consider using the product. But you do have to read the license and make sure you understand the terms and conditions.

Some non-OSI-compliant licenses are attached to software that is developed and released for academic purposes. This often occurs with the usage of university-based software development projects. Commercial users may find these licenses to be more strict.

You might also encounter "dual-licensing." One example of this is the MySQL database which is released under two licenses, one of which conforms to the OSI standards.

## Project Management

There are also project management concerns that must be taken into account when adopting open source software as part of a development project. For example, you may need to allot more time in the initial phases of the project to allow the development team time for discovery and learning. A proof of concept/prototype phase may be a good idea. This may be very important especially if the development team has never used the product in the past.

The process should take into account risk management issues associated with the maturity of the open source product and developer knowledge of the product. Many times, the risks will only be understood by the developer, who will need to help the management team understand the risks.

Another area that can contribute to project management risk is the quality of documentation for the open source product. Although a diminishing stereotype, there are still open source projects that lack documentation. Developers will be developers. The expectation that code is good enough to understand the software may indicate that a project that is still in the inception phase.

The Proof of Concept phase becomes very important in this case because the developer will need to determine the viability of the software in question. This is not to say that all open source software is immature. In fact many open source projects contain more functionality and fewer bugs than Commercial Off the Shelf (COTS)-based equivalents.

Risk management may play an increased role depending on the maturity of the product. Although not ultimately a means of determining software maturity, a critical mass of developers on a project can speak to how stable a project is, and how much support the project will have. There are open source projects (like those of the Apache Software Foundation) that have over one thousand people looking at the source code, submitting bugs, patches, and enhancements. There is a full development lifecycle that is properly defined, documented, and followed.

Because of the high level of maturity, from a risk management standpoint many organizations treat software developed by well established organizations like the Apache Software Foundation or JBoss, Inc. as they would treat COTS-based software. There are also projects that are maintained by only one person—but these may be mature, well documented projects also. Evaluate on a case-by-case basis.

## Support

Support works a little differently in the world of open source. For some projects, it is possible to purchase a support contract from either the creators of the software, or from a consulting company that is either partnered with the organization or that employs a contributing developer to the project. Other times, the only means of support include mailing lists, archives, a wiki, and documentation.

Management may see the lack of paid-for support as contributing to the overall risk of the project. The reality, however, is that having access to the developers on the project results in more expedited answers than going through the multiple levels of support one would find when contacting companies of COTS-based products. The developer may have to explain the support risks associated with an open source product and how that plays into the overall project risk matrix.

## Etiquette

There are some unwritten rules a developer must follow when communicating with the open source community. First, the developer is expected to have read through the license, documentation, any available wikis, and most importantly the mail/discussion archives. It is the responsibility of the developer to seek out any answers using this medium before asking a question that was previously asked. If an error is found, it may have come up in the past and was addressed in the mail, newsgroup, or discussion forum archives.

Developers are also expected to install the software, deploy any examples available, and attempt to develop a "Hello World." If the developer is not successful at installing the software, then the expectation is that the developer will proceed to read through the installer/build scripts and any necessary source code, make modifications to the environment or build scripts and attempt to re-install the software.

If still not successful, the developer can then send mail or post a message to the community annotating what was performed, and what error is occurring. Sending a message stating that something does not work without having done the proper due diligence to figure it out for yourself is poor etiquette.

Although most people in the open source community do respond with advice regardless of whether the message was deemed intelligent, it is in your best interest to do your homework first.

## Conclusion

The benefits associated with open source software can be realized as long as there is a plan to mitigate any risks that are part of using open source products. The ubiquity of open source software is increasing. Whether used as a small tool for software development, or as the core infrastructure for a COTS-based product, open source software will be incorporated into many specifics of an increasing number of software projects this year. Being aware of the specifics and issues around open source software will help the software developer plan appropriately for project success.

###

Johanan Lancaon is a software architect consultant living in Southern California. He has worked on projects in a variety of industries, including bioinformatics, e-commerce, finance/credit, and product development. He has been responsible for the architecture, development, and project management of high transaction software systems in the worlds of both J2EE and LAMP. He can be reached at [javageek AT gmail DOT com](mailto:javageek AT gmail DOT com).