In Perspective :

Open Source

in a Validated System

# Table of Contents

[Table of Contents 1](#_Toc467747233)

[The Selection of an Open Source Provider 2](#_Toc467747234)

[The Management of Open Source Libraries 4](#_Toc467747235)

[The Validation of Applications that Utilize Open Source 4](#_Toc467747236)

[Open Source Development Tools and Utilities 5](#_Toc467747237)

[The Benefit of and the Argument for Open Source 5](#_Toc467747238)

# 

# The Selection of an Open Source Provider

A key to selecting any third party component or library is assessing the reliability and the longevity of the vendor. This applies to open source as well as commercial third party software.

For smaller open source projects the vendor selection process it is actually less critical than for commercial vendors due to the fact that we are able to obtain and manage all the source code for the project as if it was custom in house developed software. For larger open source projects it is less realistic that we would be able to maintain the project overtime in the event that the open source provider disappeared or ceased to develop the component or library. For this reason it is important to only utilize larger open source projects from well established providers such as IBM, Sun, Apache, and many others which are listed in Appendix A.

The rigor required during the selection process should be determined based upon:

1. How critical the open source software component or library is to an overall project. The goal is to ascertain if by not utilizing a particular open source component or library a project could not be successfully implemented due to complexity, time constraints, or budgetary constraints.
2. How extensively the open source software component or library is utilized within an overall project. The goal is to ascertain the level of ongoing dependency upon the open source component or library and whether it is acceptable.
3. How complex the open source software is and whether the development team would be able to manage it on their own in the future in the event that the provider ceased to support it. As a side note, this level of flexibility is not available with most commercial software since code is generally not provided.

This is essentially a risk analysis step that should be undertaken prior to utilizing an open source component or library. The outcome of the risk analysis could be a simple rating such as:

1. High : The component or library is critical to a project, it will be used extensively within the project, and it is either complex or relatively large compared to the size of the project.
2. Medium : The component or library is critical to a project, it will be used moderately within the project, and is not complex or large relative to the size of the project.
3. Low : The component or library is useful for a project, it will be used sparingly within a project, and it is not complex or large relative to the size of the project.

Based upon the level of risk certain criteria should be defined that identify the type of provider assessment that is required. In general this assessment could include:

1. Determination whether the provider is a well-established open source provider with a proven track record and wide industry support. To simplify this analysis the company could maintain a running list of open source providers that meet this standard. Example organizations would include Apache, IBM, Sun, etc...
2. The level of support available. This is typically directly proportional to the level of utilization with in the industry since support is typically provided through forums, mailing lists, and frequently asked question lists.
3. The quality of the documentation available for the component or library.
4. The rigor of the change control utilized by the provider. This can typically be ascertained by directly viewing the provider’s cvs repository, examining the component or libraries change log with each release, examine how the provider packages components for distribution.
5. In addition, prior to adopting an open source component or library the project development team should perform a code inspection on the component or library. This can be done at a general level to ensure that the code is written to a standard equivalent to that of in house custom developed software. This would include the level of source code documentation, the coding idioms, and the decomposition, essentially a look over to ensure that a competent developer wrote the software. In addition more rigorous code inspection could be performed to ensure that the software does not attempt to do anything malicious such as open network sockets and send a scan of a hard drive. These things are generally easy to uncover. If you have a string-handling library you can do a quick scan on it to see if is utilizing any network related resources which it obviously should not. The level of rigor used during this process should be determined based upon the risk analysis.
6. Assess the level of unit tests that were performed by the open source provider. If these tests are not adequate for a particular project we could develop additional unit tests to satisfy our needs. The unit tests for the component or library should be executed to ensure that they run successfully before adoption of the component or library.

# The Management of Open Source Libraries

The management of open source components and libraries should adhere to the same requirements for custom in house developed software. The source code, binaries, and documentation should be maintained under strict internal change control. In house changes to the software should be clearly identified within the source code and appropriately tagged within the utilized change control system. When new releases of open source software become available they should be appropriately merged with any in-house changes and tested to ensure that everything still works. This is essentially the same process that is followed for updates to commercial third party software components.

Each project should document how they manage open source components within the project’s change control system, whether it is maintained within the same code branch as the project code or whether it is maintained under a separate branch. It would be useful to develop guidelines for this so that all projects organize third party code in a similar manner in order to make it easier to determine whether projects are in compliance. At minimum projects must document clearly the manner in which third party code is managed within their source control system.

# The Validation of Applications that Utilize Open Source

The validation process for applications that utilize third party components from open source providers should be the same as the process for applications that utilize third party components from commercial vendors. The goal of validation is to ensure that an application operates correctly according to the user requirements. The application is a black box that accepts user inputs, processes the inputs, potentially interacts with additional systems, and produces output. The validation process ensures that the application accepts the expected inputs and generates the expected outputs and interacts in appropriate and expected ways with the external systems. In addition, validation ensures that the application properly deals with invalid inputs, and provides the security that it claims to provide.

Should validation be more rigorous for applications that utilize open source compared to applications that only utilize commercial components? Absolutely not, validation should be rigorous and complete for both. Regardless of the quality of a software component, once it is plugged into an application it becomes as stable or unstable as the application itself. There is no guarantee that an application utilizes a commercial component or an open source component in a correct manner. The purpose of validation is to ensure through an independent means that the application actually does what the users requested and what the developers claim. How it accomplishes a task internally is out of the scope of the validation process – that is the architectural process. Whether an application multiplies numbers through a multiplication operator or a shift operator is not relevant to the validation process, what is relevant is that when you multiply 2 times 3 the result is 6 and that when you multiply 2 times –1 the result is –2; the outcome of validation is a determination of whether a system operates correctly and if not then a report indicating under what circumstances the system operated in an unexpected manner.

# Open Source Development Tools and Utilities

Developers today, Java developers in particular, have a vast and wonderful toolset available to them and most of it is available free of charge. These tools include the almost essential build tool called ANT, the ubiquitously utilized unit testing framework called JUnit, and the tremendously powerful full featured integrated development environments eclipse and netbeans to name only two of the best out of dozens. You may be asking yourself but are there not commercial alternatives to these tools. In most cases, but not all, yes there are. However, take JUnit the unit-testing framework as an example, there are commercial alternatives, but they cannot keep pace with the rate of change and with the armies of developers that have ported JUnit to over 20 development languages and written dozens of extremely useful add-ons. This is at the heart of the open source success. Commercial companies write software for proven technologies that have a large and potentially profitable market. Open source on the other hand is typically developed by developers to solve problems they are experiencing or by organization interested in seeing standards flourish. The reality today is that open source usually leads the direction of the commercial software market – not the other way around.

From the point of view of a developer this is exciting stuff, but it is not relevant to the issue of using open source components within a validated application. The tools and utilities utilized during development are not part of the deployed validated application any more than the hammer a builder uses to construct a house is part of the finished structure.

# The Benefit of and the Argument for Open Source

The open source development tools and utilities are not the real power or benefit of open source within the context of developing applications. The real power and benefit is the Holy Grail the software industry has been and remains in search of – component reuse. Open source provides a vast library of tried and tested components that have been used to solve real problems on real projects. The reuse of these components not only makes it cheaper and quicker to develop enterprise software but in truth it can be said that they make it possible. A company like Wyeth simply does not have the internal resources to develop a compliant xml parser and to keep it compliant and neither does Oracle or bea the maker of Weblogic; all of these organizations rely on the open source xml parser from Apache. There are many other less complex components used by applications that individually could possibly be developed in house.

However, the reality of developing an enterprise software application today is that it requires the combination of a large number of different technologies. To develop all of these technologies in house is impossible. To purchase all of these technologies from third party vendors is also very problematic because very often they only solve a piece of the problem and due to their proprietary nature do not easily integrate with other components and libraries. Open source components and libraries fill this gap in an extremely flexible and cost effective manner.