DON'T FORGET ABOUT THE "PEOPLE COSTS"

TCO CALCULATION FOR THE SELECTION OF ENTERPRISE MIDDLEWARE

By Francis Pouatcha, technical lead at adorsys

Note: Francis Poutcha is a distinguished enterprise architect with nine years of experience developing and implementing mission-critical projects in global 500 enterprises. He recently led a team of Java consultants on a five-year project to deploy and maintain a comprehensive middleware solution in a leading German bank's credit system.

TABLE OF CONTENTS

ABSTRACT

Most white papers and webinars available on the Internet about choosing enterprise middleware focus the selection process around criteria such as licensing costs, technical coverage of long lists of specified functionalities, and vendor viability. While these criteria may have been suitable in the past, the experience of many enterprise architects leading J2EE projects over the past few years suggest that they are not enough.

What's missing? People costs. "People costs" include a wide range of factors that promote—or prevent—IT staff productivity : availability of information around the application server platform or development environment, simplicity of the middleware platform, accessibility to the software, and quality of technical support, among others. These factors should gain more weight in the strategic selection process because they impact total cost of ownership as much as, and in some cases more than, license and ongoing support costs.

Francis Pouatcha and his team led a five-year project to implement JBoss Enterprise Middleware into the main infrastructure of a leading German bank's commercial credit service. This white paper examines in depth the associated costs of this project and how these costs should be considered in any similar project where manpower is involved during the technology's conception, development, implementation, and ongoing use. In addition, a methodology is proposed for estimating the "people costs" for BEA WebLogic, IBM WebSphere, and JBoss Enterprise Middleware, as a result of the experience gained in a real-world customer environment.

On average, findings indicate that when compared to JBoss Enterprise Middleware, BEA WebLogic requires 18% higher staffing costs to implement and 7% higher staffing costs for ongoing maintenance, while IBM WebSphere requires 23% higher staffing costs to implement and 9% higher staffing costs to maintain.

TRADITIONAL SELECTION CRITERIA

LICENSING AND SUPPORT COSTS

Due to a significant standardization level of J2EE (now Java EE) technologies, cost structures of application server platforms have thoroughly changed in the last five years. Standardization processes have led to development of a number of freely available components that are reused either unchanged or in lightly modified form by middleware vendors. Vendors who chose to develop their own found that the result was higher costs and lower quality, with more bugs than those that are commonly developed and used by all other application server platforms. For example, it is rare to find a middleware vendor that develops a proper XML parser or an application server platform with absolutely no Apache software components integrated. As such, the price paid for using an application server platform is defined by the vendor's cost of designing a suitable architecture for component assembly; the cost of maintaining commonly developed parts; the cost of assembling parts into a well-tested, fully functional application platform; and finally, the cost needed to build a sales infrastructure and maintain relationship to customers (users).

Incumbent vendors may leverage the inherent risk and uncertainty in changing application platforms to increase their profit margins--essentially profiting from vendor lock-in. Commercial vendors will leverage these margins as reserves if challenged by an open source vendor, at times significantly discounting or even giving products away at a zero license cost. Based on this experience, it seems prudent to stop emphasizing initial licensing costs as the sole cost factor. A mixed model that combines license, support, and ongoing people/productivity costs seems to be more suitable for the today's market. A vendor can choose

to waive license fees and even significantly discount support fees to remain competitive in an account, especially if there are other hardware or software products or professional services in play. But, aA vendor, however, cannot adjust the relative number of developer resources required to achieve the same level of productivity on a given application server platform when compared to other application server platforms.

DEGREE OF TECHNICAL COVERAGE OF STANDARDS

The degree of compliance to standards is mostly relevant for products and components intended to be ported or deployed on different application server platforms. For the need of an organization developing software components for their own internal use, the common set of standards provided by most leading application server platforms is mostly sufficient. This is due to the intensive re-use of commonly developed, standards-based, freely available, and well-tested parts in the assembly-oriented production process of application server platforms. For the modest need of a consuming company, degree of compliance to standards is no longer an issue. Standards compliance is not just about which standards are supported but also which proprietary functionality is layered on top of the standards. These proprietary hooks can increase vendor lock-in, even on standards-compliant platforms.

VENDOR VIABILITY

Vendor viability has traditionally played a big role in the selection process. This was reinforced as freely available components started gaining the attention of decision makers. From a risk management perspective, it is still reasonable to consider vendor viability as an important issue since any disruption in the vendor's business might have an impact on the customer's investment. That said, "vendor viability" as a decision criterion must be modified in a modern decision making processes.

The more accurate way to measure viability for enterprise middleware is not to look at the market capitalization of the vendor, but rather the viability of the components and code that the vendor provides and the quality of the services that vendor wraps around the code. Even closed source vendors that offer products via a traditional license model are leveraging open source components or releasing into open source some of their previously closed source components. Since open source code will live on in a community even if a vendor disappears, it seems prudent to measure not only to what extent a particular platform leverages open source, but how that platform is licensed. Do you actually have access to the source code? If so, the viability of the platform would be considered quite high.

Additionally, it is important to consider the viability of the vendor for the life span of the target system only. The probability of a vendor's existence in 10 years will not matter much if the company is considering a system for the next five years only, or if they are planning to move into a different system five years from now. In fact, from a practical standpoint, it is more sound to put limits on the duration of the investment given the speed of innovation driving the actual software market.

SUGGESTED SELECTION CRITERIA

To ensure the efficient allocation of resources (time and money), the process of selecting a J2EE platform for the development, deployment, and operation of business applications must take into account the ever-present aspect of money spent in developing and maintaining an application built and deployed on a particular platform. This change of focus resulted from the observations of several system integrators on the full life cycle of a number of J2EE-based business applications in the financial sector. Engineers surveyed had never met a J2EE project where licensing costs were higher than developer staffing costs. In some cases, developer staffing exceeded license costs by 10:1. Because of this factor, it is important to take a closer look at the level of resources spent in developing the software. Even though this variable may depend greatly on the way requirements are managed, focus can still be put on the amount of effort and time needed by a developer to implement a given requirement. This effort can vary greatly depending on the development environment and target application server platform. Below are productivity factors that should be considered when selecting an application server platform.

STARTUP TIME OF THE APPLICATION SERVER PLATFORM

Startup time includes the time required to install and configure an instance of the application server platform. This time can vary from minutes to hours or even days, depending on the platform. In many projects, developers use Tomcat to develop their web component wherever possible because of the fast round trip it provides. It's possible to get Tomcat up and running in milliseconds. This observation extends itself to the development of EJB components. This means the longer the startup time of an application server, the longer the time and the higher the effort needed to implement a requirement. Note that this is independent of how experienced the developer is in working on that platform. Paragraph Style: WP Bulleted List 1. This style automatically inserts the bullet and takes care of all the indenting for you. This style automatically inserts the bullet and takes care of all the indenting for you.

AVAILABILITY OF INFORMATION AROUND THE APPLICATION SERVER PLATFORM

"Information" refers to everything that could help a programmer solve issues associated with the implementation of a use case on that platform. This includes freely available sample source code of applications built on that platform; academic tutorials; forums and discussions on eventual bugs and workarounds; the ability to leverage the platform on/with specific environments like Eclipse or Ant/Maven; and availability of the source code of the application server platform. These are all worthy input resources for shortening development time. The more information developers have, the faster they can become productive. A great example is the recent productivity boost Java developers experienced with the Remember the productivity of Java programmers gaining a boost with the release of the JDK source code.Paragraph Style: WP Bulleted List (2nd Paragraph). Use this if an item in a list has more than one paragraph. It allows a hard return without an automatic bullet.

FLUIDITY OF SOFTWARE DEVELOPMENT RESOURCES

The more accessible a platform, the easier it is to find people owning the knowledge for developing on that platform. Most of them start using easily accessible platforms from the university to complete their diploma theses and homework. Some will use them to make a tutorial or exercise the technology. So if an organization adopts a platform that is light, simple, and accessible to the public, it will achieve a better fluidity of developers and save a lot of learning time while solving vacancies and job fluctuation issues.

EMBEDDING APPLICATION SERVER RUNTIME INTO THE DEVELOPMENT ENVIRONMENT

Having seen firsthand many IT projects naturally moving from EJB1.1/2.1, to Spring/Hibernate, to embedded EJB3 over the years, experienced system integrators know that embedding the runtime into the development environment has a particularly significant positive impact on development time and on the quality of resulting applications. They have learned that more higher quality test cases are the inevitable result of providing developers with an easier way of testing the application. These systems integrators also agree on the fact that the cost of maintaining and extending a business application highly depends on the quantity and quality of unit tests provided while developing that business application.

With EJB1.1 and EJB2.1, J2EE development was tightly coupled to the code-package-deploy-test life cycle, leading to very long round trips for development or maintenance of single use cases. In these situations, server startup time mentioned above was a decisive criterion of productivity. The lighter and faster the server, the less time a developer needs to redeploy and test a use case. For this reason, many developers have opted to use JBoss in the development environment to write and test applications, despite the fact that their data centers were running BEA WebLogic or IBM WebSphere.

In an effort to reduce development time and complexity when an embedded EJB container is not available, some developers may attempt to directly access the database from the web tier without the clean transactional and security features provided by EJB containers to decouple unit testing from server round trip time. For projects that require container transaction and security, developers tend to write the component in plain old Java objects (POJOs) out of the container and then embed them into EJB to use the container infrastructure. This, unfortunately, is not always possible due to the necessity of using container services to implement the business logic (e.g.:=, for implementation of exception cases, container will decide whether or not to roll back a transaction).

More recently, frameworks such as Spring and Hibernate became more mature and could provide developers with usable alternatives to the heavy EJB2.1 development and deployment model. It suddenly became possible to use a well-structured framework to develop enterprise applications with centered business logic and proven concepts for data access. One of the reasons many systems integrators and developers voted for the adoption of Spring was the ability to develop and test your code without leaving your source editor or being bound to the restart and deployment cycle of an application server. Although hot deployment could be considered from time to time, it has never been a true alternative for a lighter development round trip. Despite the lightness of alternative frameworks like Spring, leaving the standard-based, well-structured EJB development model is inevitably painful, due to the lack of services such as remoting, management, and declarative role-based access control, or the lack of a clear component and role model that allow the assembly of a large number of components into a huge business application.

The eventual maturing of EJB3 and JPA was well-received by the community, as this could retain the EJB standardization and component-oriented spirit while following the widely adopted best practices of the Spring/Hibernate world. But where there was a framework providing a well-standardized set of EJB services and a POJO-like simplicity, it was unfortunately without a clear statement on how to develop and test a use case without leaving the development environment. This is where the initiative of some vendors like Red Hat, offering an embedded version of their JBoss EJB container, could be used to test the developed EJB under the same conditions as a later stage of an EJB container, but with faster redeployment time that neutralizes the time lost for deploying the bean in an EJB container. Without this capability, developers would be far less inclined to leave Spring/Hibernate and move back to EJB3/JPA.

CALCULATING THE "PEOPLE COST" COMPONENT OF TCO

Data gathered from developers with experience in developing business applications on Red Hat JBoss, BEA WebLogic, and IBM WebSphere, (mostly contractors who have been involved in numerous projects from different companies), showed that to implement the same business application, JBoss Enterprise Application Platform required 18% less developer time than BEA WebLogic and 23% less developer time than IBM WebSphere during mainstream development (learning curve included).

JBoss also required 7% less developer resources to maintain the application than BEA WebLogic and 9% less than IBM WebSphere. The additional maintenance costs incurred by BEA WebLogic and IBM WebSphere come from the higher startup time of these servers, the fewer number of reference EJB applications built on these servers, the lack of availability of their source code, and the resulting complexity associated with developing and testing on these platforms. The methodology for these results is as follows.

Developing a typical mid-size application requires an average of 32 developers in the first year for the mainstream development, 18 developers in the second and third years for maintenance and business logic extensions, 11 developers in the fourth year for maintenance, and six developers during the fifth year for managing end of life and migration issues. The mix of developers who are employees vs. contractors is considered in the calculations below.

For the mainstream development in the first year, the average project needed 32 developers actively working 200 days during the year. Among them are eight employees on payroll and 24 contractors. The average daily rate of a standard contracting developer is Y Euro and the average salary of a developer on payroll in the company is 1.2X (X plus 20% ancillary labor costs).

An additional detail to be considered will be the VAT unrecoverable by the institution building the business application for proper use. For a development project taking place in Germany for example, a VAT of 16% for 2003-2006 and 19% for 2007 must be accounted for. Given these considerations, development staffing costs for a mid-sized application in a typical enterprise are detailed in Table 1.

TABLE 1

COST POSITION	AMOUNT ON JBOSS	ADDITIONAL EXPENSE IF BEA WL +18%	ADDITIONAL EXPENSE IF IBM WS + 23%
Employees on payroll (8 employees x 1.2X)	9.6X	1.73X	2.21X
Contractors (24 contractors * Y(Euro/ day) * 200 days * 1.16 (VAT))	5568.0Y	1002.24Y	1280.64Y
Opportunity costs		1,73X + 1002,24Y	2,21X + 1280.64Y

This result can be interpreted as BEA requiring 1.73 more developers on payroll and 1,002.24 more contractors days to build the same application during the mainstream development phase in year 1 (2003).

Maintenance and business logic extensions in 2004 and 2005 needed an average of 26 developers (eight on payroll and 18 contractors). For maintenance (even considering the experience of developers on those platforms), there was still a need for about 7% more developer resources for BEA and about 9% for IBM. Again, the main reasons for this are startup time, lack of examples, and heaviness of the platform. For a typical enterprise in 2004 and 2005, the staffing expenditure for maintenance is detailed in Table 2.

TABLE 2

COST POSITION	AMOUNT ON JBOSS	ADDITIONAL EXPENSE IF BEA WL +7%	ADDITIONAL EXPENSE IF IBM WS + 9%
Employees on payroll (8 x 1,2X)	9.6X	0.67X	0.86X
Contractors (18 contractors * Y(Euro/ day) * 200 days * 1.16 (VAT))	4176.0Y	292.32Y	375.84Y
Opportunity costs		0.67X + 292.32Y	0.86X + 375.84Y

In 2006, while business logic activity slowed, staffing requirements were reduced to 6 employees on payroll and 5 contractors. Staffing costs for 2006 are detailed in Table 3.

TABLE 3.

COST POSITION	AMOUNT ON JBOSS	ADDITIONAL EXPENSE IF BEA WL +7%	ADDITIONAL EXPENSE IF IBM WS + 9%
Employees on payroll (6 x 1,2X)	7.2X	0.5X	0.65X
Contractors (5 contractors * Y (Euro/ day) * 200 days * 1,16 (VAT)	1160.0Y	81.2Y	104.4Y
Opportunity costs		0.5X + 81.2Y	0.65X + 104.4Y

For 2007, the average enterprise is building a new version of the business application so they have a feature freeze for the old application. But the company still has to maintain the old application because of new regulations and competition. Developer requirements in 2007 are three employees and three contractors. Staffing costs for 2007 are detailed in Table 4.

TABLE 4.

COST POSITION	AMOUNT ON JBOSS	ADDITIONAL EXPENSE IF BEA WL +7%	ADDITIONAL EXPENSE IF IBM WS + 9%
Employees on payroll (3 x 1.2X)	3.6X	0.25X	0.32X
Contractors (3 contractors * Y(Euro/day) * 200 days * 1,16 (VAT))	714.0Y	49.98Y	64.26Y
Opportunity costs		0.25X + 49.98Y	0.32X + 64.26Y

Staffing costs for development and maintenance to be added to the licensing/subscription based TCO comparison over the life cycle of the business application are summarized in Table 5.

TABLE 5.

	JBOSS	BEA	IBM
2003	9.6X + 5568.0Y	11.33 X + 6570.24Y	11.81X + 6848.64Y
2004	9.6X + 4176.0Y	10.27 X + 4468.32Y	10.46X + 4551.84Y
2005	9.6X + 4176.0Y	10.27 X + 4468.32Y	10.46X + 4551.84Y
2006	7.2X + 1160.0Y	7.70 X + 1241.20Y	7.85X + 1264.40Y
2007	3.6X + 714.0Y	3.85 X + 763.98Y	3.92X + 778.26Y
Sum	39.6X + 15794.0Y	43.43 X + 17512.06Y	44.51X + 17994.98Y
Opportunity costs		3.83 X + 1718.06Y	4.91X + 2200.98Y

To summarize the results of this methodology, if an on-staff developer costs 40,000 EUR per year, and the daily rate for a contract developer is 400 EUR per day, then the average developer staffing cost for a mid-sized application in an average enterprise would be 840,424 EUR more on BEA WebLogic and 1,076,792 EUR more on IBM WebSphere than on JBoss Enterprise Application Platform.

Note that this example is still not a complete TCO calculation. It shows only the people costs that should be added to the license/support and/or subscription fees to get an accurate estimate of total cost of ownership.

SUMMARY

The intention of this paper is to highlight the fact that "people costs" are significantly impacted by the attributes of an application server platform that promote or prevent IT staff productivity. Even though traditional criteria such as license costs and provider viability might still play a big role in the selection process, people costs are a critical factor in accurately determining the total cost of ownership for an application server platform. In addition, a vendor's market capitalization is no longer the dominating factor in the assessment of provider viability. The adoption of common components and the rise of open source business models require viability to be redefined in terms of the software and the services wrapped around it, not the vendor.

In fact, the cost of developer productivity should not only be included in a total cost of ownership analysis; it should be given considerable weight, since experience shows that developer staffing costs always outweigh license and support costs--in some cases as much as 10:1. Factors that significantly impact developer productivity and software quality (and therefore developer staffing costs) include the lightness of the application server, faster startup time for round trip development, and embedded testing in the development environment.

Among the three application server platforms observed, JBoss Enterprise Application Platform offered the greatest opportunity for developer productivity. Its embedded EJB3 container reduces development round-trip time. Its unrestricted access of the source code also makes comprehension of the overall container architecture easier. Additionally, easier access to the core JBoss application server has lead to the proliferation of thousands of sample applications also freely available on the Internet.

In conclusion, the observations in this paper support the benefits of releasing middleware components and platforms into open source, thus reducing barriers to accessing server software and corresponding source code. It encourages each viable middleware provider, whether they currently distribute via open source or perpetual license models, to consider providing developers with productivity tools such as an embedded container that allows for easy and fast development and testing of business applications in an environment equal, or very close to, the production runtime environment.

People costs are intrinsically related to IT staff productivity. The availability of information around the application server platform, development environment, simplicity of the middleware platform, accessibility to the software, and quality of technical support directly impacts the total cost of ownership as much as, and in some cases more than, license and ongoing support costs. Money saved on developer time could be reinvested into a backlog of application projects that, if completed, would provide or help maintain a sustainable competitive advantage for the company.