



Return on Investment for Composite Applications and Service Oriented Architectures: A Model for Financial Success and Enterprise Efficiency

Joshua Greenbaum
Enterprise Applications Consulting
Fall 2005



EAC

2303 Spaulding Avenue

Berkeley CA 94703

tel 510.540.8655

fax 510.540.7354

josh@eaconsult.com

www.eaconsult.com

Table of Contents

Introduction: The ROI Challenge for Service Oriented Architectures and
Composite Applications page 1

ROI in SOA & Composite Applications: The Four Key Domains page 3

Building on the Past: Using Applications Development and Enterprise Applications to
Define Return on Investment for SOA page 10

Conclusion: Towards a Lower Cost of Innovation and a Higher Return on Technology
and Business page 14

Introduction: The ROI Challenge for Composite Applications and Service Oriented Architectures

Service oriented architectures and composite applications hold out the promise for a brave new world of applications development, deployment, and reuse that many proponents believe will usher in unprecedented levels of return on investment (ROI) for a domain that has long suffered from cost overruns and excessive, often unjustified expenditures. The ability to lower the cost of integration while improving the leveragability of key software and business process assets are only a few of the reasons why the ROI of service-oriented architectures and composite applications is thought to herald a new economic reality for IT and business development. Ease of use and lower training costs, lower cost of deployment, faster time to market, improved business requirement matching, and better multi-channel deployment are among the myriad reasons these technologies are so eagerly awaited by business and IT managers alike.

While the theory seems sound, however, understanding how that theory actually works in the real world has been difficult. In particular, a well-developed understanding of the return on investment for service-oriented architectures and composite applications has been a complex undertaking. This is due in part to the dearth of comprehensive, historical data on which to base any such model. For the most part, composite applications and the service oriented architectures (SOA) on which they are built more often exist as pilot projects than as full-blown production systems, and even those rare production-quality systems that do exist are too new for use in understanding critical issues to the ROI equation, such as reusability and redeployment.

The other problem militating against building a comprehensive ROI model for service-oriented architectures and composite applications is the need to separate the general ROI that results from any well-designed application and the specific or incremental ROI that obtains from a well-designed composite application built on a services architecture. Indeed, much of the current ROI data available in the market fail to make this distinction, and therefore render many of these results useless as tools on which to base real-world analysis.

Composite applications draw on existing data and services in order to build unique, new functionality that can deliver business value to a company, very often in a flexible deployment mode that includes desktop, mobile, rich client, and other interfaces. A service-oriented architecture is the platform on which a composite application is developed and deployed, and in which the components or services that are available for reuse can be managed.

Nonetheless, it is possible to understand the ROI for SOA by exploring possible parameters for analysis as well as by looking at data and analysis used to cost-justify older applications and business development processes. EAC's review of ROI models for applications development and enterprise applications integration (EAI) offers some compelling insights into ROI for SOA. While the ROI model for service-oriented architectures and composite applications that is presented in this report is theoretical, it can help identify key areas – such as the hand-coding of business services and their reusability – in which SOA and composite applications have a distinct advantage over previous development and deployment options. EAC's research shows that customers can expect an overall ROI of 13 to 35 percent above what can be gained from standard application development methods. The overall ROI with respect to EAI is even higher, and is the result of the extreme disparity in costs between the two models. Both findings augur well for what the ROI of service oriented architectures and composite applications will look like in real-world, production-quality applications.

This report is organized into three parts. The first is a comprehensive discussion of the four key areas where users and developers can expect an ROI on their SOA and composite applications endeavors. The second section uses ROI models for applications development reuse and enterprise applications integration to demonstrate the incremental ROI that an SOA can provide. The report concludes with a discussion of the combined benefit of tangible and intangible factors on understanding the overall ROI of service architecture.

ROI in Composite Applications & SOA: The Four Key Domains

Before we look at the historical ROI of applications development and EAI, it's important to clearly define composite applications and SOA as well as understand the broad basis of their potential ROI. Composite applications are those applications that draw on existing data and services in order to build unique, new functionality that can deliver business value to a company, very often in a flexible deployment mode that includes desktop, mobile, rich client, and other interfaces. A services-oriented architecture is the platform on which a composite application is developed and deployed, and in which the components or services that are available for reuse can be managed.

EAC's research shows that there are four key domains where end-user organizations can expect to see a return from this new model of software development and deployment. While there is some degree of overlap between the four different domains, it's important to see that the breadth of potential ROI spans a broad range of functional areas and stakeholders in the enterprise. This broad-based impact is one of the most compelling aspects to service-oriented architectures and composite applications, and provides the grounds for an enterprise-wide ROI that is unprecedented in the world of applications development and deployment.

The four domains of a service oriented architecture/composite applications ROI are:

- ◆ Development ROI
- ◆ Deployment/Maintenance ROI
- ◆ IT Extension ROI
- ◆ User/Expert ROI

DEVELOPMENT ROI PARAMETERS

The ROI for applications development has three basic parameters – services abstraction, service integration abstraction, and rapid service assembly – each of which provides a significant ROI individually. Together, they form a compelling theoretical basis for ROI in the service architecture/composite applications world. (See Table I.)

Services and Meta-services Abstraction

The ability to “wrapper” existing business processes into web services is a key part of the ROI of service oriented architectures and composite applications. While their use is an essential part of the ROI for rapid development (see below), it is important to look at services abstraction as a separate ROI factor. The act of automating business processes into software code is the *raison d'être* of enterprise software. Making that automation reusable and more directly tied to business requirements is the value-add that SOA provides, in addition to facilitating the programming of the business processes themselves. This value-add is greatly extended by the ability to make changes and upgrades to individual services – changing the business logic in a financial service to reflect changes in the tax code, for example – and then leveraging that change across all the component applications that access the particular financial service.

The abstraction of services that describe the interaction between other services is another key part of the ROI for service architectures. These “meta-services” – some SOA vendors refer to them as “business services” – often define company-specific, or purpose-specific, functionality between other generic services, such as a company-specific process for invoice reconciliation. The unique value-add in this case – a service that links invoicing, customer relationship management, and order management components, for example – must be part of the service oriented architecture and be available for use in composite applications development.

Table I: Development & Deployment ROI Parameters and Savings Potential

ROI PARAMETER	SAVINGS POTENTIAL
Convert Business Process and Logic into Reusable Web Services	<ul style="list-style-type: none"> • Greater Alignment with Business Requirements • Less Error Due to Discontinuity between Business Processes and Applications Development
Reusable Meta-Services or Business Services	<ul style="list-style-type: none"> • Faster Development • Less Error Due to Reuse of Existing Services
Assembly of Composite Applications from Services	<ul style="list-style-type: none"> • Reduction in Time and Error Rate Due to Manual Coding
Meta-Services or Business Services Abstraction	<ul style="list-style-type: none"> • Reduction in Recoding • Increase in Reusability of Established Applications Integration Services
Multi-Channel Deployment	<ul style="list-style-type: none"> • Lower Cost of Redeployment • Savings in Coding for Multiple Channels
Lower Upgrade and Redeployment Costs	<ul style="list-style-type: none"> • Faster Time to Market • Lower Overall Development Costs
Single Repository for Storing Services and Meta-Services	<ul style="list-style-type: none"> • Faster Time to Development and Deployment
Rapid Deployment	<ul style="list-style-type: none"> • Faster Time to Market • Lower Overall IT Costs
Reuse of Existing Services	<ul style="list-style-type: none"> • Lower Maintenance and Service Costs

Rapid Development

The ability to more rapidly develop highly functional composite applications is a hallmark of the ROI of these new technologies. The abstraction of complex business logic, application integration, or data integration into reusable web services means that hand-coding can be replaced with a component assembly model that is significantly faster and more efficient. This component assembly process does not just involve applications or business logic: A well-designed SOA will enable the development of services that are truly appropriate for composite applications development, resulting in a refinement over many existing Web services that are neither reusable nor robust enough. A well-designed SOA will also provide support for componentized application integration services or business services as well. This means that the onerous and expensive task of applications integration can be simplified and accomplished much less expensively by a less technically-savvy programmer or developer.

Rapid development in an SOA is also realized by significantly lowering the error rate in applications development, thereby lowering the testing and debugging costs as well. This is accomplished by the reuse of existing software assets – an SOA allows developers to test once and deploy broadly with a minimum of secondary testing.

Multi-channel deployment of composite applications is also more cost-effective in an SOA. Because interfaces and platform-specific implementation can be rendered as services, the overhead of deploying the same composite application to a desktop, telephone, or PDA, for example, is significantly reduced and can be accomplished without the use of highly technical programming resources. Automating the generation of user interfaces to match the form factor of a particular deployment channel makes it much easier to assemble a collection of business services into a composite application.

Finally, the cost of applications redeployment, revision, and upgrade is also much lower. Required changes need to be made only to the affected components or services, as opposed to making changes to the entire code base. Once made, those changes can then be automatically deployed to the components applications that use the services by the SOA with a minimum of overhead and error.

DEPLOYMENT/MAINTENANCE ROI PARAMETERS

Faster deployment and lower maintenance costs are also important ROI components for an SOA. Rapid deployment in part comes from the fact that core business services used by composite applications are already part of the existing applications infrastructure. Likewise, issues like security, scalability, and reliability have also been “baked” into the core services and need not be addressed every time a service is reused. Thus, assuming a business service has already been componentized, its deployment is technically a “non-event” when used by a composite application.

Similarly, maintenance costs are lower in an SOA environment due to this same reuse factor. Assuming that the underlying software architecture and hardware are sufficiently scalable, there is no additional maintenance cost for the use of existing componentized business processes in new composite applications.

Deployment in a multi-channel environment, as noted above, is also less costly in an SOA environment, due to the fact that multi-channel deployment is itself a service made available by SOA. This makes the cost of deploying an existing application in a new channel – such as a PDA or phone – negligible.

IT EXTENSION ROI PARAMETERS

The ROI benefits for the IT department have some overlap with the above ROI parameters, but it’s important to distinguish the direct impact of service oriented architectures and composite applications on the IT department. IT departments can see a significant ROI for SOA deployments through the extension and repurposing of existing applications and services, as noted above. This extension and repurposing not only takes place without a corresponding increase in maintenance and deployment costs – which are often charged to the IT department – but it also has the effect of extending the ROI of the existing enterprise applications that provide the business services used by the composite applications. (See Table II.)

Legacy applications are particularly open to this repurposing ROI, as SOA and composite applications can enable the creation of new business logic, processes, and user interfaces that extend the value of legacy applications. A further ROI can be obtained by application portfolio rationalization that helps consolidate service/process overlap and cost-justify the sunsetting of

redundant applications. This portfolio rationalization, of course, is not limited to legacy applications, and can be applied across the board to legacy and more modern enterprise applications alike. An improved alignment of IT services and business needs can also derive from SOA and from the deployment of composite applications that are faster to develop and fit more closely with business requirements. Finally, IT departments can also expect a better return on their human capital costs from this new development and deployment model, as the lower development costs, faster development times, and code-free development process increase productivity and/or lower personnel costs.

Table II: IT Extension ROI Parameters and Expected Savings

ROI PARAMETER	SAVINGS POTENTIAL
Repurpose Legacy Applications and their Services	<ul style="list-style-type: none"> • Extended ROI of Existing Applications Portfolio
Portfolio Rationalization	<ul style="list-style-type: none"> • Lower License, Maintenance, and Support Costs • Lower Overall IT Costs
Better Alignment with Business Requirements	<ul style="list-style-type: none"> • Increased Completion Rate • Better Overall Use of IT Services
More Rapid and Efficient Applications Development and Deployment, More Efficient Applications Management	<ul style="list-style-type: none"> • Lower Human Capital Costs • Lower IT Services Costs

END-USER/BUSINESS ROI PARAMETERS

Development of an end-user ROI from composite applications and service-oriented architectures starts with two basic components. The first is the overall efficiency and productivity enabled by the automation of non-automated business processes: This is the main justification for building composite applications, and it has the added potential benefit of improving end-user morale by reducing the drudgery that often accompanies non-automated end-user processes.

End-user ROI can also come from the lower training costs that result from componentized applications, particularly those that replace poorly designed legacy applications and the convoluted “work-arounds” that typify many poorly automated or manual business processes. End-user benefits can also accrue from the development of new composite applications that provide a single coordinated user interface that masks an underlying heterogeneous applications environment. While this lower ROI is dependent on good design practices, the ability to improve upon the interfaces and human/machine interactions of legacy and outdated ERP applications can be the basis of significantly lower training costs.

Table III: End-User and Business Benefit ROI Parameters and Expected Savings

ROI Parameter	Savings Potential
Efficiencies from Automation of Business Processes	<ul style="list-style-type: none"> • Faster Time to Market • Improved Resource Utilization
Lower Training Costs	<ul style="list-style-type: none"> • Faster End-user Uptake • Lower User Error Rates • Lower IT Help Desk Utilization • Improved Employee Morale
Employee Productivity Improvement	<ul style="list-style-type: none"> • Lower Personnel Costs • Higher Productivity
Improved Applications Functionality	<ul style="list-style-type: none"> • Higher Productivity • Improved Competitiveness • Improved Customer/Partner Relations • Lower Error Rates
Faster Applications Development	<ul style="list-style-type: none"> • Improve Competitiveness • Improved Business Agility
Improved IT and Business Alignment	<ul style="list-style-type: none"> • Faster Time to Market • Lower Overall Development Costs • Improved Competitiveness

A number of additional ROI parameters can be drawn from this starting point. (See Table III, above.) The improvements in overall efficiency and productivity can give all employees a more productive work environment. For example, knowledge workers have better access to strategic and tactical information, while other employees can see significant reductions in redundancy, information re-keying, and other inefficient processes. Overall accuracy can also be significantly improved using well-designed composite applications: The reuse in a composite application of existing processes that have already been proven efficient means fewer points of failure, particularly when compared to custom-coded applications. Lower training costs have the additional benefit of improving employee morale and employee acceptance of new technology.

There are also some end-user ROI parameters that overlap with larger enterprise-wide ROI parameters. As stated above, time to market for new processes and applications can be improved with well-designed composite applications, resulting in more highly differentiated business strategies and improved competitiveness. Strategies and the processes that support them can be developed and implemented more quickly, resulting in a more agile response to business change. And overall, the development and deployment process associated with composite applications can help align business and IT in a more effective and efficient embrace.

Building on the Past: Using Applications Development and EAI to Define Return on Investment for SOA

While EAC has identified a broad base of factors on which an ROI model for composite applications in the context of an SOA can be built, the newness of the technology makes it difficult to build a comprehensive ROI model based on real-world data. A look at well-developed ROI models for older development and deployment technologies, however, can point the way towards understanding how these new technologies can provide a significantly greater ROI. While the historical models we will consider here don't cover the complete universe of ROI parameters as outlined in the previous sections, the incremental ROI that the new technologies can provide over the two historical models discussed in this section should more than justify the adoption of service-oriented architectures and composite applications. EAC expects that, as the industry-wide experience with service-oriented architectures and composite applications broadens, and the data on the other ROI factors outlined above become available, the complete picture will show an even greater potential ROI than discussed here.

The two models that will be used for this comparative view are ROI models for software reuse and enterprise applications integration. The software reuse model, developed by Jeffrey Poulin of Lockheed Martin Systems Integration, is particularly useful in understanding how simple incremental increases in reusability and lower development costs and error rates that come from service-oriented architectures and composite applications development can bring significant savings to already efficient development processes.

The second model, while less comprehensive, is useful for understanding not only the overall deployment costs of service-oriented architectures and composite applications, but also for understanding these costs relative to an important predecessor technology, enterprise applications integration, or EAI. The model to be used here was originally developed by BEA Systems to describe the ROI for its applications integration solution. It has a particularly useful role here in highlighting how much more cost-effective service-oriented architectures and composite applications can be in delivering ROI for applications integration development and deployment.

It's important to note that one artifact of using historical models of ROI is the cost difference between older development and deployment technologies and newer technologies such as SOA. The overall project costs in the original models cited below are significantly higher than what can be expected in an SOA development environment for two reasons. The first is that the "big bang", multi-million dollar projects of the past are very much passé, and average development project costs are often one tenth as large today as they were when Poulin and BEA were proposing these models. The second reason is that the efficiencies to be gained from SOA and composite applications are so significant that EAC believes that development can genuinely cost orders of magnitude less with SOA than with previous models. EAC's calculations of costs relative to SOA and composite applications below reflect the lower overall development cost structure that can be expected from this new model.

Software Development ROI and Service-Oriented Architectures

The model that Poulin has developed is both comprehensive and complex (see Appendix I), but the basics are relatively straightforward. In a standard software development environment, there are two key parameters that can be used to define the difference between standard software development and SOA software development: the amount of software that can be reused, and the cost of developing software for reuse.

If we change key parameters in Poulin's model to fit the SOA world, the relative ROI of SOA is dramatically lower than in the pre-SOA software development world. Whereas in a typical pre-SOA project cited by Poulin the cost savings from code reuse would be \$165,000 for 30,000 lines of code, a similar SOA-based project – using relatively conservative estimates – would yield a cost savings from re-use of \$187,500. Bearing in mind that these estimates are based on a relatively small project, the overall savings would grow significantly for enterprise-class projects with hundreds of thousands of lines of code.

*SOA software development can yield from
13 percent to 35 percent lower costs
as compared to non-SOA development.*

Indeed, looking at another example in Poulin's research of a much larger project – one that encompasses a total of 1.2 million lines of code, with 240,000 lines available for reuse – and changing the key parameters to account for the savings in an SOA environment, Poulin's equation yields a proportionally larger ROI of 35 percent.

EAI Development/Deployment and SOA/Composite Applications

The ROI model presented by BEA in 2003 for EAI project development and deployment is much less comprehensive than Poulin's model, but it is illustrative of the potential for the significant savings available to companies that use SOA's integration capabilities as a replacement for EAI technology. (See Table IV.) It also highlights the enormous disparity between EAI-driven mega-projects and the smaller cost models associated with SOA. (A more comprehensive analysis of the BEA model relative to SOA costs can be found in Appendix II).

Table IV: Relative Costs of EAI vs. SOA Project

	EAI*	SOA**
License Cost	\$6,000,000	\$150,000 - \$300,000
Implementation	\$400,000	\$50,000
Configure and Build Reusable Components/Processes	\$1,600,000	\$160,000
Total Software and Architecture Cost	\$8,075,000	\$585,000
Integration Development	\$23,463,992	\$100,000 - \$200,000
Software Maintenance	\$2,170,800/year	\$162,000/year

* Source: BEA

** Source: EAC estimates

As the figures in the above table indicate, a head-to-head comparison between an older EAI model and SOA yields such differences in costs as to make them almost incomparable. This exposes as much the failings of EAI as it does the promise of SOA: the cost of EAI, in particular relative to SOA, was astronomically high, more so considering the high rate of failure and customer dissatisfaction. At a minimum, exposing the relative cost structures of EAI and SOA can help companies judge how far their SOA investments can go relative to EAI.

This difference in cost structure between EAI and SOA becomes even more significant when one weighs the relative value of the integration architecture built by EAI and the composite applications that are enabled by the development of an SOA. In the EAI model, none of the benefits from developing and deploying composite applications can be applied to the enormous expense incurred. Key composite application parameters like reuse, improved user interface, and faster development and deployment have no analog in EAI, and are therefore completely missing from an ROI comparison between the two methods. The resulting relative cost-effectiveness of SOA over EAI is even starker when the value of composite applications is added to the picture.

Conclusion: Towards a Lower Cost of Innovation and a Higher Return on Technology and Business Services

The bottom line for software development and deployment is that the prospects for a significant return on investment using service oriented architectures and composite applications is excellent, given this relatively conservative rendering of the differences between the older models of software development and deployment and what can be expected in a service oriented architecture environment. Whether a company is developing the business services that are to become part of the composite applications repertoire, wrapping legacy systems as services, or developing and deploying composite applications in a service oriented architecture, the potential cost savings over existing or previous methodologies will be significant.

Perhaps an even more important ROI than the hard numbers that can be derived from the analysis in this report is the “soft” or more intangible ROI that can be obtained from deploying a service oriented architecture and developing composite applications. The most succinct way to look at these more intangible issues is in terms of the degree to which innovation, competitiveness, and operational effectiveness can be improved using an SOA and composite applications approach.

If – as the data in this report suggest – software development costs can be reduced by 13 to 35 percent, and integration costs, including deployment, can be reduced even further, then it stands to reason that a similar degree of savings in terms of time and productivity can also be obtained from these new technologies. A savings of 13 to 35 percent in productivity and time to innovation for new software products and services could have an important impact on not just profitability, but customer and partner satisfaction as well as the overall competitive profile of a company. The fact that this can take place in a lower cost, more productive environment means that the benefits that accrue directly to the business side can be shared with IT and operations, as well as with customers, business partners, and other stakeholders. Individual users can be more productive, and entire departments and lines of business can benefit from the savings that come from an improved software development and deployment environment.

Enterprise Applications Consulting believes that the ROI discussed in this report can be delivered by an SOA solution, and that the incremental value of this ROI will increase as customers’ experiences with SOA development grow and the quantity of service-enabled software in the customers’ portfolios grow as well. This potential for even greater ROI over time is perhaps the

best possible endorsement of service oriented architectures and composite applications. Without exception, every IT and business manager would do well to consider how these new technologies can bring a similar ROI into their organizations.

Appendix One: Exploring SOA ROI Using the Poulin Model

According to Poulin, only 20 percent of the total cost of development can be directed towards reuse, a factor Poulin calls RCR. In the pre-object-oriented development world, 20 percent would represent an impressive number, but in the world of service oriented architectures and component applications, EAC believes that 80 percent is a more accurate figure. This ability to reuse the majority of the software developed by an organization is one of the key attributes of SOA development, and while the number will vary greatly from one development organization to another, EAC believes that the early adopters will see this or an even greater degree of reuse simply because initial SOA development will target precisely those applications and business processes that have the greatest reuse potential.

The other essential metric from Poulin is the relative cost of writing software for reuse, or RCWR. In the world of standard software development, according to Poulin's extensive research, the RCWR for most software is 50 percent, meaning that it costs an extra 50 percent to code for reuse. This relative cost uptake disappears in the SOA world, where the cost of coding for reuse is essentially zero.

What these changes mean in terms of ROI can be seen by looking at Poulin's ROI equations. Poulin takes a project in which 30,000 lines of code are intended for reuse and comes up with what he calls the Total Reuse Cost Avoidance – the overall savings possible – using the following equation:

Total Reuse Cost Avoidance = Development Cost Avoidance (DCA) + Service Cost Avoidance (SCA)

Development Cost Avoidance, i.e. the amount a development organization can save through reuse, is calculated in the following manner:

DCA = Lines of Code x Relative Reusability (calculated from the RCR) x Cost of each New Line of Code

If we use Poulin's original numbers (with the exception of his cost per new line of code, which Poulin calculated at \$100/line, and which EAC estimates is more accurately \$5/line), the cost avoidance or net savings in a "pre-SOA" environment that promises 20 percent reuse cost recovery for 30,000 lines of code looks like the following:

$$DCA = 30,000 \times (1 - 0.2 \text{ (RCR)}) \times \$5 = \$120,000$$

The other part of Poulin's Total Reuse Cost equation is Service Cost Avoidance – the cost to be avoided by reusing code that has been already tested and debugged. Service cost is calculated in the following way:

$$SCA = \text{Lines of Code} \times \text{Error Rate per 1000} \times \text{Cost to Fix Each Error}$$

In an SOA environment, Service Cost Avoidance differs less than Development Cost Avoidance due to the fact that, while the error rate with SOA is substantially lower, the value of avoiding errors remains constant between the new and old development method. So in this case, we leave Poulin's original number of 1.5 errors per thousand constant, but change his cost of fixing errors, which was tagged at \$10,000 per error – a number EAC believes is more accurately expressed as \$1,000 per error. This yields the following Service Cost Avoidance for the same 30,000 lines of code:

$$SCA = 30,000 \times (1.5/1000) \times \$1,000 = \$45,000$$

This results in the total cost avoided, or saved, from reuse in a pre-SOA environment of:

$$DCA + SCA = \$120,000 + \$45,000 = \$165,000$$

Now if we substitute EAC's figures of 80 percent cost reuse, while maintaining the same service cost avoidance, Poulin's DCA equation yields a significantly different result:

$$DCA = 30,000 \times (1 - 0.05) \text{ (RCR at 80 percent cost reuse)} \times \$5 = \$142,000$$

As the SCA, \$45,000, remains the same, the total cost avoided in an SOA environment is the following:

$$\text{DCA} + \text{SCA} = \$142,000 + \$45,000 = \$187,500$$

This represents a total ROI improvement of 13 percent. Obviously, while the difference is only \$22,000, this is a rate of savings per 30,000 lines of code. A project that would reuse a larger percent of code – EAC believes that it is reasonable to expect that over 80 percent of code developed for SOA-based projects would be intended for reuse – would yield significant additional savings.

Appendix II: Exploring SOA ROI using the BEA Model

BEA's model proposes the following EAI cost structure for a large, enterprise-wide project:

Integration Development	\$13,388,992
Hardware Cost	\$2,000,000
Architecture Development	\$2,000,000
Software Cost	\$6,075,000
<hr/>	
Total Deployment Cost	\$23,463,992

BEA further breaks down the total of \$8,075,000 in architecture development and software cost expenses in the following way:

Configure/Build reusable components and processes	\$1,600,000
Implementation	\$400,000
Enterprise Software License	\$6,000,000
Other Software	\$75,000
<hr/>	
Total Architecture and Software Cost	\$8,075,000

BEA then calculates a total cost of operations in the following manner:

Integration Maintenance (3 years)	\$2,475,336
Operational Support (3 years)	\$1,575,000
Software Maintenance Fees	\$2,170,800
<hr/>	
Total Cost of Operations	\$6,221,136

Summing total cost of operations with total deployment costs, BEA's model yields a total integration cost of \$29,685,128. These are admittedly astronomically high numbers by today's standards of development, but they accurately reflect how the world of applications development and its cost structure has moved in only two years.

Thus, finding the SOA savings in this model is relatively easy, and the results are significant. Let's start with architecture and software costs. The largest single cost, the enterprise software license, is singularly lower in an SOA environment. A comparable SOA could cost \$150,000 to \$300,000, an incredible savings over BEA's cost structure. Implementation would also be significantly lower, at approximately \$50,000 for an enterprise-wide implementation. Configuring and building reusable components would also be significantly lower in an SOA environment. Based on EAC's research, the SOA figure would be \$160,000, which brings the total architecture and software cost down from \$8,075,000 using EAI to \$585,000 to SOA.

Next is integration development. Again, the much lower cost structure of SOA makes a comparison with the EAI world look off the charts. The overall lower cost of SOA architecture and software means that the scale of integration development in the BEA model is vastly outsized compared to an SOA model: EAC estimates that integration development in an SOA project like the one described above would cost approximately \$100,000 – \$200,000, not the \$23,463,992 a BEA customer could have expected to pay for a big bang integration project.

BEA's total cost of operations calculations are also subject to significant change in an SOA environment. EAC estimates that integration maintenance and operational support would be similarly lower, and reflective of the enormous disparity in cost structures between the two approaches – despite the potential for very similar results. For example, software maintenance on a \$300,000 software license, at a rate of 18 percent over three years, would cost \$162,000, not the \$2,170,800 in the EAI model.