



Understanding the Service Lifecycle within a SOA: Design Time

by Quinton Wall
10/04/2006

Abstract

Service Oriented Architecture (SOA) presents an architecture approach that relies on decomposing business processes and lower level activities into standards-based services. These services may be fine grained, course grained, presentation-centric, data-centric, or any number of other permutations. The ability to effectively manage the lifecycle of services is fundamental to achieving success within a SOA initiative. These discussions shall be divided into two articles focusing on design-time and run-time aspects of the lifecycle, respectively. This first article covers the design-time phases in the service lifecycle.

Traditional Application Development

To understand the need for service lifecycle management, it is important to recognize some of the paradigm shifts that SOA introduces into an organization. Many authors and analysts (Groves 2005, Dwyer 2006) have indicated that SOA requires closer alignment of business and IT. This in itself is often a large shift within the organization's operations. Another is the shift from development and deployment of applications to discrete services.

With regard to IT projects, an application may be defined as a collection of tasks designed to address the needs of a user or category of users. For example, a foreign exchange application may contain tasks and functionality to support looking up trades, defining an instrument, and submitting the order to a trade blotter. These functions are designed to target a category of user loosely defined as foreign exchange traders. Traditionally, an application was developed under a project or initiative within the organization. Project managers and analysts would define needs and functionality required, determine the level of effort, and identify the resources required and some controlling factors such as cost, scope, and time. Here is where some of the concerns with traditional application development lifecycles begin.

Very often requirements are *baked into* these applications at functional definition time. Agile programming methodologies have attempted to address this concern by iterating the requirements definition throughout the build process. This approach has alleviated the concern to some extent, but the rules and processing behavior may still be embedded within the application. Service lifecycle discussions will not directly address this issue but the composition of services through a SOA can provide a way of decoupling this logic. In addition, due to the tight coupling within traditional applications and the need to combine many functions within a deliverable, run-time governance of tasks is often quite difficult. Many of the decisions relating to security, quality of service, and so on are made at design time rather than at run-time, therefore greatly reducing the flexibility of the application to adapt to changing business needs.

Of greater concern to traditional application development projects are the long development and provisioning times. It is not unusual for applications to take six to twelve months before being placed into production. I have been involved in many projects that have taken a number of years before being production ready. Quicker time to market is a critical aspect of SOA's appeal. Regarding service lifecycle discussions, it is important from the perspective that traditional application development processes often resulted in attempts to squeeze as much functionality into a project scope as possible, often more than is realistic! The result of this additional scope is one of two outcomes: reduced quality of the release, or

project delays. Neither of these two scenarios is ideal for business or IT. Gaining an understanding of service lifecycle management directly addresses this problem.

With the rise of popularity and interest in SOA as a method for decomposing applications into services which can be shared across organizations it is not surprising that SOA principles are being leveraged in an attempt to address many of the negative aspects of traditional application development lifecycle. To date SOA adoption has shown considerable benefits in addressing these concerns but it also introduces a number of new requirements around lifecycle management, service management and organizational awareness. This article attempts to identify issues and best practices around what I will call the Shared Service Lifecycle (SSLC).

The Shared Service Lifecycle

Many organizations and analysts have used the term *business service lifecycle* to describe the lifecycle associated with SOA services. I prefer to use the term *shared service lifecycle*, or SSLC. Utilizing the term *business* within the definition signifies that I am talking only about those services that represent business processes (these are often composite services), not foundation services such as those that provide information and access to physical data sources and security services. By substituting *business* with *shared*, you are better equipped to recognize that service lifecycle management is directly affected by its relationship with other services in the enterprise, some of these being composite business services.

Figure 1 depicts a typical iterative SSLC divided into design-time and run-time aspects. This article will focus on the design-time aspects with a subsequent article focusing on the run-time phases of the SSLC.

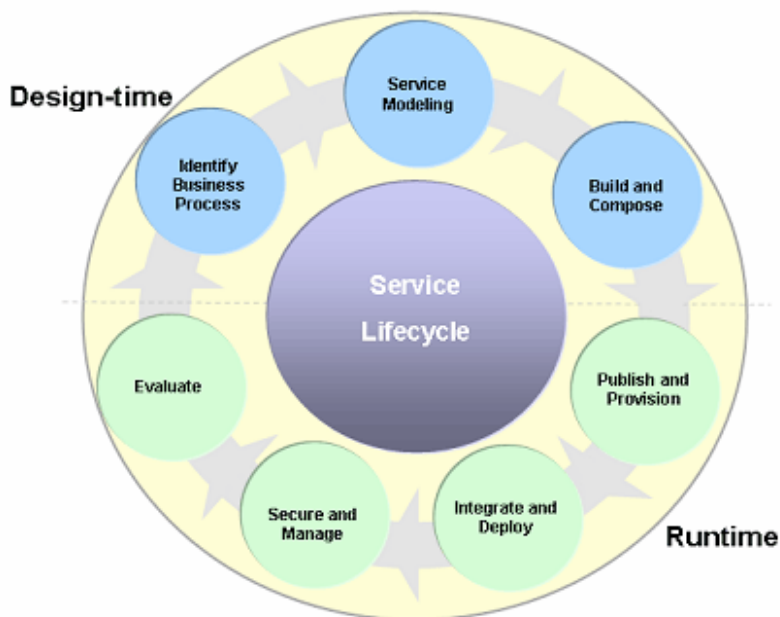


Figure 1: The design and run-time phases of a shared service lifecycle

Pages: [1](#), [2](#), [3](#)

[Next Page »](#)