41. How Should Services Be Identified or Specified to Maximize Reuse?

A key tenet of understanding SOA is the focus on getting the organization to reuse versus a focus on the programmer to reuse. Robert L. Glass, software engineer, professor, and author, wrote in his book, Facts and Fallacies of Software Engineering, that reuse in coarse-grained or large components remains mostly an unsolved problem even though everyone agrees it's important and desirable. The problem, he asserts, is that when reusable modules are built, they then have to do something that matches a large set of needs in a wide variety of programs. Robert's articles on reuse state that minimal reuse exists because simply not that many software components can be reused. This is why the focus should be on what can be reused at the business or organizational level.

Getting an organization to reuse translates to sharing business functionality across processes or workflows, within the enterprise and with partners. It requires that sharable functionality be engineered in a manner that allows sharing: Build once and share regardless of the consumer platform. This engineering requires the identification and realization of services. The identification phase in an SOA project includes not just the identification of services, but also the identification of processes, information, rules, and components. Our experience indicates that it is a best practice to utilize a set of complementary service identification techniques. Relying on a single technique tends
to create either an incomplete set of services or services without the necessary granularity to effectuate reuse and flexibility. A single technique often introduces information entropy early in the development life cycle, often to be remedied at greater cost later in the service life cycle because it entails greater efforts of service refactoring to eliminate redundancies or make the service coarser for higher business value. In addition, this often leads to the failure to identify service dependencies early on, which impacts release planning and, ultimately, project delivery.

When designing a service portfolio, you should use a combination of service identification techniques to cast a more complete net and catch the necessary services required to support a business. Some projects might choose to lead with information, whereas others choose a business process focus. Each of the techniques can be used to start the process of service identification and then use the subsequent ones to work in concert from different angles: process, information, top down, bottom up, exploring the commonality and variations across processes, information, rules, policies, and events. Any technique that is not applicable or of little value on a given project should be omitted or its usage minimized.

Services are optimally identified using three complementary techniques that provide a balance between tactical imperatives and strategic vision:

- **Goal service modeling** looks at business opportunities, strategy, and business goals to both confirm and validate that candidate services have been identified, which fulfill goals and enable the business strategy.

- **Domain decomposition** focuses on business process modeling, rules, information, and potential variability of services.

- **Asset analysis** addresses the reality that businesses have accumulated legacy systems and applications that must be integrated, enhanced, or leveraged. This bottom-up approach looks at the existing application portfolio and other assets that can be used in identifying candidates for service exposure. In contrast, goal service modeling combines the top-down (domain decomposition) and bottom-up (asset analysis) approaches and pulls them together into alignment.
Building and sharing common services that can be leveraged across multiple lines of business or the enterprise requires leveraging and reusing assets, services. The failure to share services and repurpose them in new ways puts an organization in the situation of re-creating something that has been done 60% to 100% the same way, and development dollars are spent needlessly re-creating functionality that already lives in the application portfolio of the enterprise. Sharing services does require change management and governance, which are addressed in Chapter 3, “Organization,” and Chapter 4, “Governance.” Sharing services requires that you identify and build the right services, which are what the three complementary techniques of service identification promote.

Instead of focusing on common services, the focus is on shared business processes and standard business processes. In turn, having shared services, multipurpose, multiconsumer services sourced from business processes, existing assets, information, and business goals affords the best opportunity to identify and build the right services that are shareable. Trying to identify common services should not be the goal. Instead, the goal should be on the identification and build out of services based on business goals and business processes using the three complementary techniques that result in the desired effect and shared and common services.

The services to provide or consume can be summarized and centralized in a service model. The service model includes a categorized list of services called the service portfolio. These services are abstracted into a layer in the architecture that decouples providers and consumers, through a service contract. The service model becomes an essential asset to promote reuse, which can be automated using a registry for search, dynamic binding, versioning, and other full life cycle governance features that facilitate reuse.

Using standard business processes and eliminating duplicate business processes can make a big difference in enabling sharing of processes. This is a primary reason business processes must be visible, understood, consolidated, and maintained for the life of the system, as a way for IT to maintain its connection to the business to continue to represent and understand evolving business needs.
Use cases have been widely used and recognized as a best practice for the capture of functional requirements. A use case captures a set of static actor-object interactions that ultimately realize the use case. These flows are most often invariable and hard-coded. This does not allow the easy recombination of functionality. Variability affects reuse and sharable services because these seemingly infinitesimal differences lie at the heart of the lack of reuse.

In contrast, in variation-oriented analysis, a service case identifies the reconfigurable choreography of a set of service operations, each a unit of functionality. This flow is not hard-coded. Instead of endeavoring to initially identify the objects that sequence the interactions, the focus is on the set of business aligned services that collectively enable the fulfillment of business goals, and the services can be recombined in unanticipated service contexts. Rather than being just an actor interacting with a system, the service is part of an ecosystem of providers and consumers with often interchangeable roles that leverage the services through policies and new combinations in ever-changing use cases.

42. **How Should the Granularity of a Service Be Determined?**

Granularity speaks to how fine-grained (small units of business functionality) or coarse-grained (large units of business functionality) a service should be engineered to solve a business need. The right granularity depends on context. Most business applications have both fine-grained and coarse-grained services in the service portfolio. Let’s look at the pros and cons of each.

Fine-grained services can cause frequent network hops, and thus overhead and inefficiency in their invocation. Coarse-grained services have less network chatter, but are rarely at the right level of detail required. They therefore have a more limited potential for reuse. A service can be modeled so that it can be understood whether a service is too fine because the number of network trips to fulfill an activity of business process is so high it makes the performance of the business process unacceptable. A service can also be modeled so that you can look at a service and determine whether it is too coarse because the
ability to make changes independent from the consumers is limited. This reinforces the position that \textit{granularity} is more of an adjective of how a service is described versus a verb and something to do to a service. Granularity is often a focus because of the need to optimize reuse and performance of the service.

The proper granularity of a service or a right service or the optimal service is not about deciding on granularity but on identifying services. If the focus is on service identification, granularity takes care of itself. So, the focus is not on the service, but on the business processes and how the service might meet the needs of multiple business processes and multiple consumers in the enterprise or line of business. The more the service can be leveraged, the more the service fulfills a key step of a business activity, the more the service helps to eliminate redundancy versus promote redundancy, and the more certain we can be that we have the proper granularity. Granularity of a service is driven by the needs of the known and anticipated consumers of the service rather than a design issue that can be determined without the proper context.

43. \textit{Should SOA Be Used Only for Custom Development Projects?}

SOA is useful and recommended for multiple development styles, not only the custom development projects. SOA can be used for several project styles, such as transformation, legacy enhancement, packaged implementation/integration, and information-based projects. Adopting SOA methods does not translate to using web services or exposing services as web services. Adopting service modeling as a basis for structuring applications has multiple advantages regardless of project type:

- \textit{Transformation projects} have goals that cannot be realized by a single project but require a program of projects. Such programs require a vision (an end state or strategy that can be fulfilled). For many organizations, SOA provides a key part of the strategy. SOA becomes the blueprint, starting with a documented vision of the end state using SOA principles and tenets. Companies looking for an application architecture or strategy for transformation consistently adopt SOA.
• **Legacy enhancement** and **legacy transformation** are synonymous terms used to describe the leveraging of existing applications and modernizing them to support new requirements. Legacy enhancement takes on many flavors, and for a lot of companies, this means taking existing application programming interfaces and converting these to service contracts or taking existing systems and wrapping them with web services. Each approach has its advantages and disadvantages, but by applying SOA and services you can extend the life of legacy systems. SOA extends the life of legacy in two ways. The primary way is the avoidance of new legacy systems. The other is using services as façades to enable access to existing legacy business functionality by new channels (e.g., mobile devices), other applications, or external partners.

• **Packaged implementation** often requires extensive integration, so SOA becomes an adoption scenario for both packaged implementations and integration projects. Proliferation of point-to-point solutions is costly to implement and change. Services can be used to integrate packages or integrate disparate systems. An **enterprise service bus** (ESB) is often installed to operate as an intermediary between systems. Services running on the ESB can be deployed and perform routing, protocol conversion, or data transformation. The ESB is the primary adoption pattern for packaged implementation projects. In addition, some companies choose not to adopt or allow the package and its implementation to drive its enterprise or line-of-business data model. That is, some organizations will have their own enterprise data model and packages (e.g., SAP) that are adopted will integrate with existing data models and not supersede the enterprise data model. The service becomes a means to integrate packages with existing systems, using information services, where the enterprise data model is separate and distinct from the data model that comes with the packaged application.

• **Information-based projects** are projects that require aggregation of data into information from many sources, and in which services become the mechanism for both aggregation and integration. For example, in a data warehousing or information analytics solution, services can be used as a means for providing aggregated information from multiple data sources.