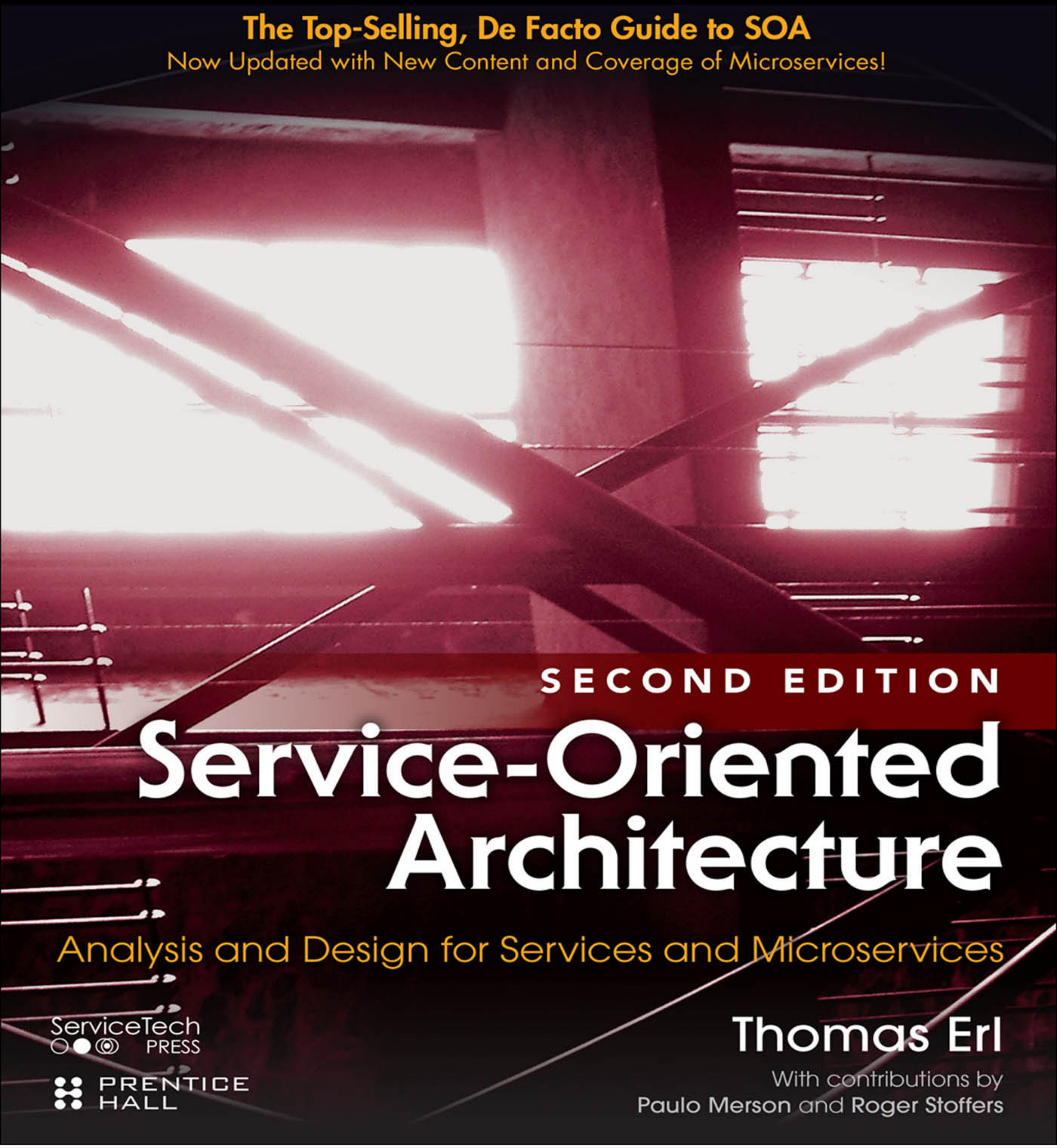
THE PRENTICE HALL SERVICE TECHNOLOGY SERIES FROM THOMAS ERL

The Top-Selling, De Facto Guide to SOA
Now Updated with New Content and Coverage of Microservices!



SECOND EDITION

Service-Oriented Architecture

Analysis and Design for Services and Microservices

Thomas Erl

With contributions by
Paulo Merson and Roger Stoffers

ServiceTech PRESS

PRENTICE HALL

Service-Oriented Architecture

This page intentionally left blank

Service-Oriented Architecture

Analysis and Design for Services and Microservices

Thomas Erl

With contributions by Paulo Merson and Roger Stoffers

PRENTICE HALL

BOSTON • COLUMBUS • INDIANAPOLIS • NEW YORK • SAN FRANCISCO
AMSTERDAM • CAPE TOWN • DUBAI • LONDON • MADRID • MILAN • MUNICH
PARIS • MONTREAL • TORONTO • DELHI • MEXICO CITY • SAO PAULO
SIDNEY • HONG KONG • SEOUL • SINGAPORE • TAIPEI • TOKYO

ServiceTech PRESS

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearsoned.com.

Visit us on the Web: informit.com/ph

Library of Congress Control Number: 2016952031

ISBN-13: 978-0-13-385858-7
ISBN-10: 0-13-385858-8

First printing: December 2016

Publisher
Mark Taub

Editor-in-Chief
Greg Wiegand

Senior Acquisitions Editor
Trina MacDonald

Managing Editor
Sandra Schroeder

Senior Project Editors
Lori Lyons
Betsy Gratner

Copyeditors
Paula Lowell
Language Logistics
Infinet Creative Group
Maria Lee
Teejay Keepence

Indexer
Cheryl Lenser

Proofreaders
Williams Woods Publishing
Abigail Gavin
Melissa Mok
Kam Chiu Mok
Shivapriya Nagaraj
Catherine Shaffer
Pamela Janice Yau
Maria Lee

Editorial Assistant
Olivia Basegio

Cover Design
Thomas Erl

Photos
Thomas Erl

Cover Compositor
Chuti Prasertsith

Compositor
Bumpy Design

Graphics
Jasper Paladino
Zuzana Cappova
Infinite Creative Group
Spencer Fruhling
Tami Young
Demian Richardson
Kan Kwai Lui
Briana Lee

Educational Content Development
Arcitura Education Inc.

To Markus, who recently joined our team
with a keen sense of curiosity and a relentless desire
to analyze and redesign even the most micro of things.
—Thomas Erl

This page intentionally left blank

Contents at a Glance

This page intentionally left blank

Contents

PART III: APPENDICES

This page intentionally left blank

Acknowledgments

This Second Edition is comprised of content from a variety of sources, including new content that reflects industry developments and revised content from other series titles. Thank you to all who helped shape what this book is comprised of, and special thanks to the following individuals who contributed new insights and new design patterns:

In alphabetical order:

- Paulo Merson
- Roger Stoffers

Reader Services

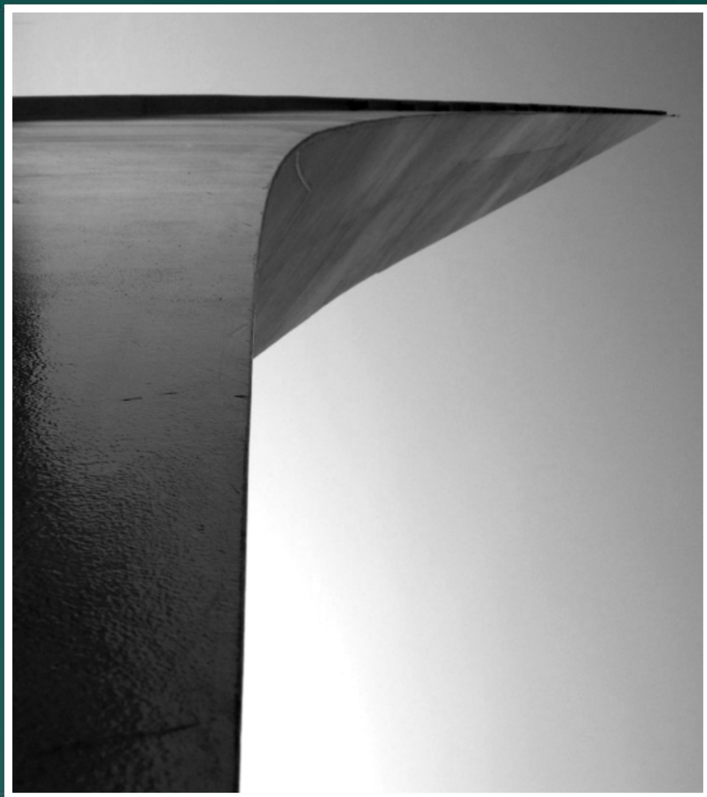
Register your copy of *Service-Oriented Architecture: Analysis and Design for Services and Microservices* at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account.* Enter the product ISBN, 9780133858587, and click Submit. Once the process is complete, you will find any available bonus content under "Registered Products."

*Be sure to check the box that you would like to hear from us in order to receive exclusive discounts on future editions of this product.

Part I

Fundamentals

This page intentionally left blank



Chapter 3

Understanding Service-Orientation

This chapter is dedicated to describing the service-orientation design paradigm, its principles, and how it compares to other design approaches.

3.1 Introduction to Service-Orientation

In the everyday world around us services are and have been commonplace for as long as civilized history has existed. Any person carrying out a distinct task in support of others is providing a service. Any group of individuals collectively performing a task in support of a larger task is also demonstrating the delivery of a service (Figure 3.1).

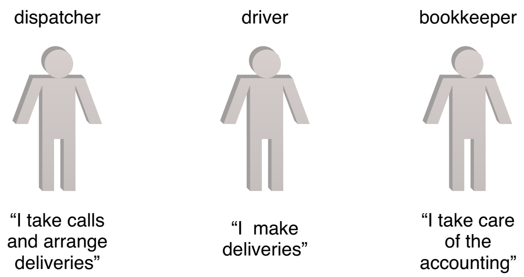


Figure 3.1
Three individuals, each capable of providing a distinct service.

Similarly, an organization that carries out tasks associated with its purpose or business is also providing a service. As long as the task or function being provided is well defined and can be relatively isolated from other associated tasks, it can be distinctly classified as a service (Figure 3.2).

Certain baseline requirements exist to enable a group of individual service providers to collaborate in order to collectively provide a larger service. Figure 3.2, for example, displays a group of employees who each provide a service for ABC Delivery. Even though each individual contributes a distinct service, for the company to function effectively, its staff also needs to have fundamental, common characteristics, such as availability, reliability, and the ability to communicate using the same language. With all of these things in place, these individuals can be composed into a productive working team. Establishing these types of baseline requirements within and across business automation solutions is a key goal of service-orientation.

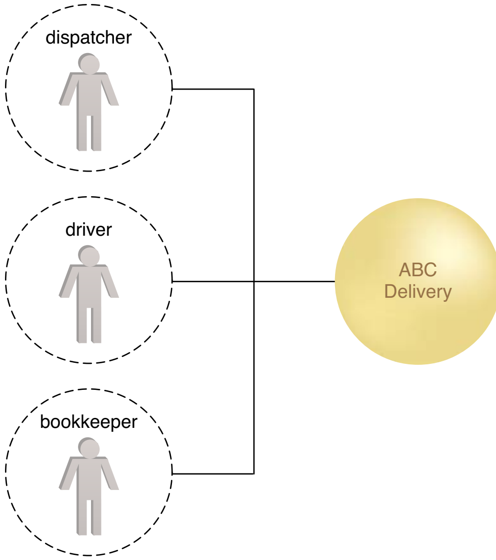


Figure 3.2
A company that employs these three people can compose their capabilities to carry out its business.

Services in Business Automation

From a general perspective, a *service* is a software program that makes its functionality available via a published API that is part of a *service contract*. Figure 3.3 shows the symbol used to depict a service (without providing any detail regarding its service contract).

Figure 3.3
The symbol used to represent an abstract service.

Different implementation technologies can be used to program and build services. The two common implementation mediums covered in this book are SOAP-based Web services (or just Web services) and RESTful services (or just REST services). Figure 3.4 shows the standard symbols used to represent service contracts in this book.

NOTE

A Web service contract is generally comprised of a WSDL definition and one or more XML Schema definitions. Services implemented as REST services are accessed via a uniform contract, such as the one provided by HTTP and Web media types. Chapters 8 and 9 provide examples of Web service and REST service contracts.

A service contract can be further comprised of human-readable documents, such as a Service Level Agreement (SLA) that describes additional quality-of-service guarantees, behaviors, and limitations. Several SLA-related requirements can also be expressed in machine-readable formats.

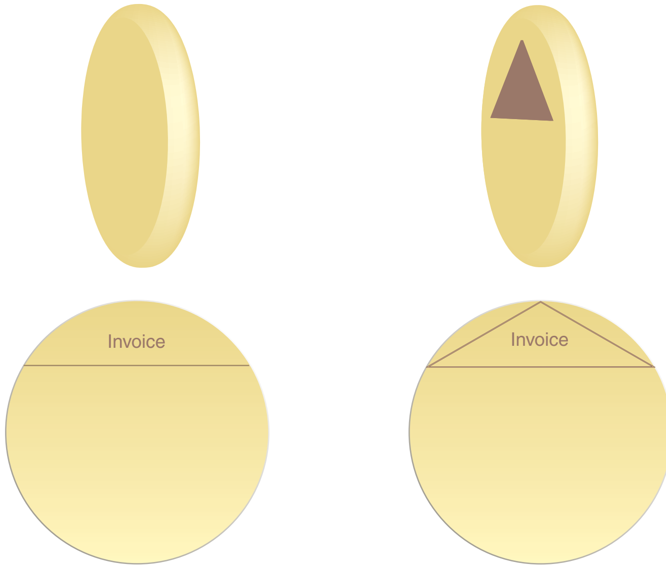


Figure 3.4
The chorded circle symbol used to display an Invoice service contract (left), and a variation of this symbol used specifically for REST service contracts (right).

Services Are Collections of Capabilities

When discussing services, it is important to remember that a single service can offer an API that provides a collection of capabilities. They are grouped together because they relate to a functional context established by the service. The functional context of the service illustrated in Figure 3.5, for example, is that of "shipment." This particular service provides a set of capabilities associated with the processing of shipments.

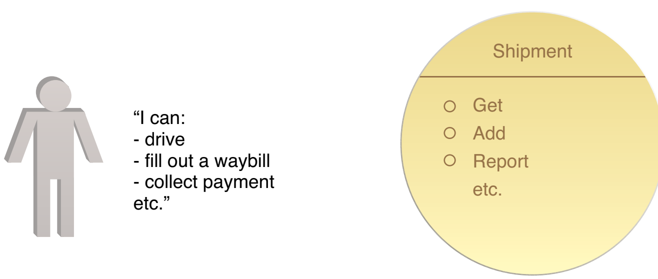


Figure 3.5
Much like a human, an automated service can provide multiple capabilities.

A service is therefore essentially a container of related capabilities. It is comprised of a body of logic designed to carry out these capabilities and a service contract that expresses which of its capabilities are made available for public invocation. When we make reference to service capabilities in this book, we are specifically focused on those that are defined as part of the service contract API.

A *service consumer* is the runtime role assumed by a software program when it accesses and invokes a service—or, more specifically, when it sends a message to a service capability expressed in the service contract. Upon receiving the request, the service begins executing logic encompassed by the invoked capability and it may or may not return a corresponding response message to the service consumer. A service consumer can be any software program capable of invoking a service via its API. A service itself may act as the consumer of another service.

AGNOSTIC VS. NON-AGNOSTIC LOGIC

The term “agnostic” originated from Greek and means “without knowledge.” Therefore, logic that is sufficiently generic so that it is not specific to (has no knowledge of) a particular parent task is classified as agnostic logic. Because knowledge that is specific to a single-purpose task is intentionally omitted, agnostic logic is considered multipurpose. Conversely, logic that is specific to (contains knowledge of) a single-purpose task is labeled as non-agnostic logic.

Another way of conceptualizing agnostic and non-agnostic logic is to focus on the extent to which the logic can be repurposed. Due to the multipurpose nature of agnostic logic, it is expected to become reusable across different contexts so that the logic, as a single software program (or service), can be used to help automate multiple business processes. Non-agnostic logic is not subject to these types of expectations. It is deliberately designed as a single-purpose software program (or service) and therefore has different characteristics and requirements. Non-agnostic logic can still be reusable, but only within the scope of its parent business process, which preserves its context as being specific to a greater, single-purpose task.

Service-Orientation as a Design Paradigm

A design paradigm is an approach to designing solution logic. When building distributed solution logic, design approaches revolve around a software engineering theory known as the "separation of concerns." In a nutshell, this theory states that a larger problem is more effectively solved when decomposed into a set of smaller problems or *concerns*. This gives us the option of partitioning solution logic into capabilities, each designed to solve an individual concern. Related capabilities can be grouped into units of solution logic.

Different design paradigms exist for distributed solution logic. What distinguishes service-orientation is the manner in which it carries out the separation of concerns and how it shapes the individual units of solution logic with specific characteristics and in support of a specific target state.

Fundamentally, service-orientation shapes suitable units of solution logic as enterprise resources that can be designed to solve immediate concerns while still remaining agnostic to the greater problem. This provides the constant opportunity to reutilize the capabilities within those units to solve other problems as well.

Applying service-orientation to a meaningful extent results in solution logic that can be safely classified as "service-oriented" and units that qualify as "services." (Chapter 5 explores in detail how the separation of concerns is carried out with service-orientation.)

Services, as part of service-oriented solutions, exist as physically independent software programs with distinct design characteristics. Each service is assigned its own distinct functional context and is comprised of a set of capabilities related to this context. A *service composition* is a coordinated aggregate of services. As explained later in the *Effects of Service-Orientation on the Enterprise* section, a composition of services (Figure 3.6) is comparable to a traditional application in that its functional scope is usually associated with the automation of a parent business process.

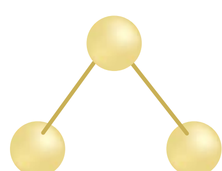


Figure 3.6

This symbol, comprised of three connected spheres, represents a service composition. Other, more detailed representations are based on the use of chorded circle symbols that illustrate which service capabilities are actually being composed.

A *service inventory* is an independently standardized and governed collection of complementary services within a boundary that represents an enterprise or a meaningful segment of an enterprise. Figure 3.7 establishes the symbol used to represent a service inventory in this book.

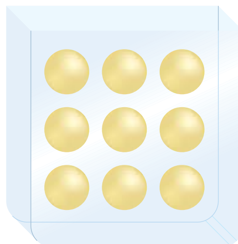


Figure 3.7
The service inventory symbol is comprised of spheres within a container.

An IT enterprise can contain or may even be comprised of a single service inventory. Alternatively, an enterprise environment can contain multiple service inventories. When an organization has multiple service inventories, this term is further qualified as *domain service inventory*.

The application of service-orientation throughout a service inventory is of paramount importance to establish a high degree of native interservice interoperability. This supports the repeated creation of effective service compositions (Figure 3.8).

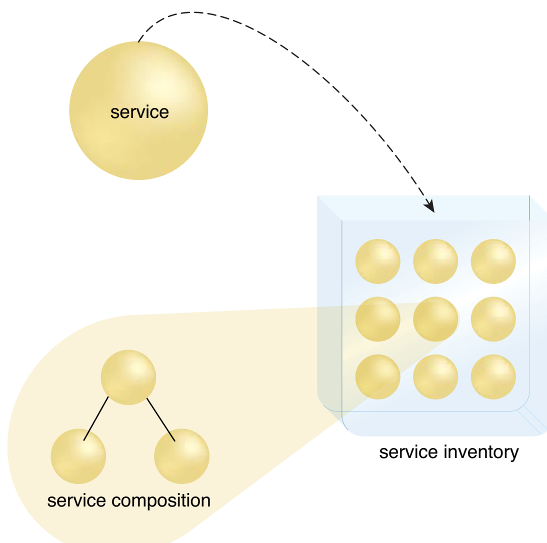


Figure 3.8
Services (top) are delivered into a service inventory (right) from which service compositions (bottom) are drawn.

Here's a brief recap of the elements of service-orientation that have been covered so far:

- *Service-oriented solution logic* is implemented as *services* and *service compositions* designed in accordance with *service-orientation*.
- A *service composition* is comprised of *services* that have been assembled to provide the functionality required to automate a specific business task or process.
- Because *service-orientation* shapes many *services* as enterprise resources, one *service* may be invoked by multiple consumer programs, each of which can involve that same *service* in a different *service composition*.
- A collection of standardized *services* can form the basis of a *service inventory* that can be independently governed within its own physical deployment environment.
- Multiple business processes can be automated by the creation of *service compositions* that draw from a pool of existing agnostic *services* that reside within a *service inventory*.

As explored in Chapter 4, service-oriented architecture is a form of technology architecture optimized in support of services, service compositions, and service inventories.

Service-Orientation Design Principles

The preceding sections have described the service-orientation paradigm at a very high level. But how exactly is this paradigm applied? It is primarily applied at the service level (Figure 3.9) via the application of the following eight design principles:

- **Standardized Service Contract (291)** – *Services within the same service inventory are in compliance with the same contract design standards.*

  Services express their purpose and capabilities via a service contract. This is perhaps the most fundamental principle in that it essentially dictates the need for service-oriented solution logic to be partitioned and distributed in a standardized manner. It also places a great deal of emphasis on the design of service contracts to ensure that the manner in which services express functionality and define data types is kept in relative alignment.

- **Service Loose Coupling (293)** – *Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment.*

  Coupling refers to a measure of dependency between two things. This principle establishes a specific type of relationship within and outside of service boundaries, with a constant emphasis on reducing ("loosening") dependencies between a service contract, its implementation, and service consumers. Service Loose Coupling (293) promotes the independent design and evolution of service logic while still guaranteeing baseline interoperability.

- **ServiceA bstraction(294)** – *Service contracts only contain essential information and information about services is limited to what is published in service contracts.*

  Abstraction ties into many aspects of service-orientation. On a fundamental level, this principle emphasizes the need to hide as much of the underlying details of a service as possible. Doing so directly enables the previously described loosely coupled relationship. Service Abstraction (294) also plays a significant role in the positioning and design of service compositions.

- **Service Reusability (295)** – *Services contain and express agnostic logic and can be positioned as reusable enterprise resources.*

  Whenever we build a service, we look for ways to make its underlying capabilities useful for more than just one purpose. Reuse is greatly emphasized with service-orientation—so much so, that it becomes a core part of the design process and it also forms the basis for key service models (as explained in Chapter 5).

- **Service Autonomy (297)** – *Services exercise a high level of control over their underlying runtime execution environment.*

  For services to carry out their capabilities consistently and reliably, their underlying solution logic needs to have a significant degree of control over its environment and resources. Service Autonomy (297) supports the extent to which other design principles can be effectively realized in real-world production environments.

- **Service Statelessness (298)** – *Services minimize resource consumption by deferring the management of state information when necessary.*

  The management of excessive state information can compromise the availability of a service as well as the predictability of its behavior. Services are therefore ideally designed to remain stateful only when required. Like Service Autonomy (297), this is another principle that focuses less on the contract and more on the design of the underlying logic.

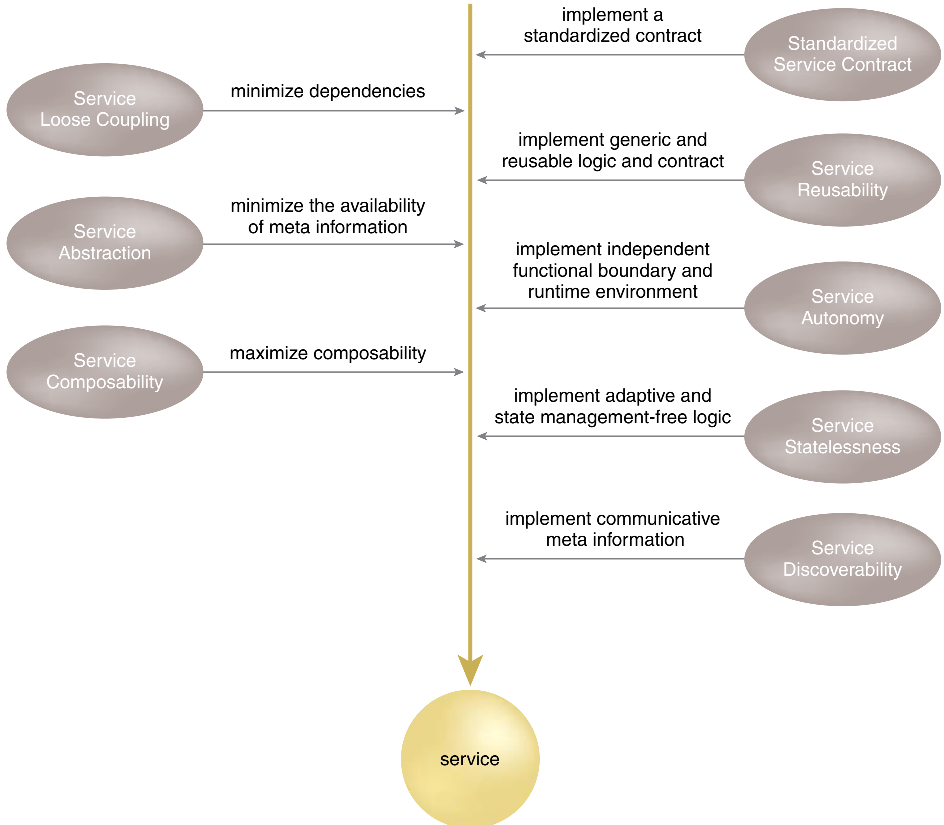


Figure 3.9
How service-orientation design principles collectively shape service design.

- **Service Discoverability (300) –** *Services are supplemented with communicative meta-data by which they can be effectively discovered and interpreted.*

  For services to be positioned as IT assets with repeatable ROI, they need to be easily identified and understood when opportunities for reuse present themselves. The service design therefore needs to take the "communications quality" of service contracts and capabilities into account, regardless of whether a discovery mechanism such as a service registry is an immediate part of the environment.

- **Service Composability (302) –** *Services are effective composition participants, regardless of the size and complexity of the composition.*

  As the sophistication of service-oriented solutions grows, so does the complexity of underlying service composition configurations. The ability to effectively compose services is a critical requirement for achieving some of the fundamental goals of service-oriented computing. Complex service compositions place demands on service design. Services are expected to be capable of participating as effective composition members, regardless of whether they need to be immediately enlisted in a composition.

SOA PATTERNS

Service-orientation principles are closely related to SOA patterns. Note how each pattern profile table in Appendix C contains a field dedicated to showing related design principles.

3.2 Problems Solved by Service-Orientation

To best appreciate why service-orientation emerged and how it is intended to improve the design of automation systems, we need to compare before and after perspectives. By studying some of the common issues that have historically plagued IT we can begin to understand the solutions proposed by this design paradigm.

Silo-based Application Architecture

In the world of business, delivering solutions capable of automating the execution of business tasks makes a great deal of sense. Over the course of IT's history, the majority of such solutions have been created with a common approach of identifying the business tasks to be automated, defining their business requirements, and then building the corresponding solution logic (Figure 3.10).

This has been an accepted and proven approach to achieving tangible business benefits through the use of technology and has been successful at providing a relatively predictable return on investment (Figure 3.11).

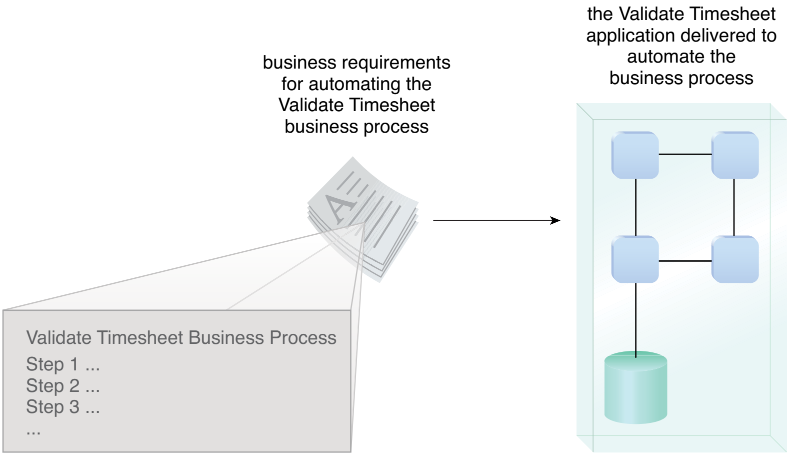


Figure 3.10
A ratio of one application for each new set of automation requirements has been common.

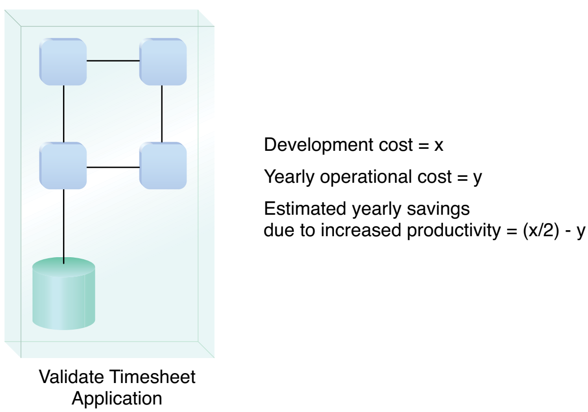


Figure 3.11
A sample formula for calculating ROI is based on a predetermined investment with a predictable return.

The ability to gain any further value from these applications is usually inhibited because their capabilities are tied to specific business requirements and processes (some of which will even have a limited lifespan). When new requirements and processes come our way we are forced to either make significant changes to what we already have or build a new application altogether.

In the latter case, although repeatedly building "disposable applications" is not the perfect approach, it has proven itself as a legitimate means of automating business. Let's explore some of the lessons learned by first focusing on the positive.

- Solutions can be built efficiently because they only need to be concerned with the fulfillment of a narrow set of requirements associated with a limited set of business processes.
- The business analysis effort involved with defining the process to be automated is straightforward. Analysts are focused only on one process at a time and therefore only concern themselves with the business entities and domains associated with that one process.
- Solution designs are tactically focused. Although complex and sophisticated automation solutions are sometimes required, the sole purpose of each is to automate just one or a specific set of business processes. This predefined functional scope simplifies the overall solution design as well as the underlying application architecture.
- The project delivery lifecycle for each solution is streamlined and relatively predictable. Although IT projects are notorious for being complex endeavors, riddled with unforeseen challenges, when the delivery scope is well-defined (and doesn't change), the process and execution of the delivery phases have a good chance of being carried out as expected.
- Building new systems from the ground up allows organizations to take advantage of the latest technology advancements. The IT marketplace progresses every year to the extent that we fully expect technology we use to build solution logic today to be different and better tomorrow. As a result, organizations that repeatedly build disposable applications can leverage the latest technology innovations with each new project.

These and other common characteristics of traditional solution delivery provide a good indication as to why this approach has been so popular. Despite its acceptance, though, it has become evident that there is still much room for improvement.

It Can Be Highly Wasteful

The creation of new solution logic in a given enterprise commonly results in a significant amount of redundant functionality (Figure 3.12). The effort and expense required to construct this logic is therefore also redundant.

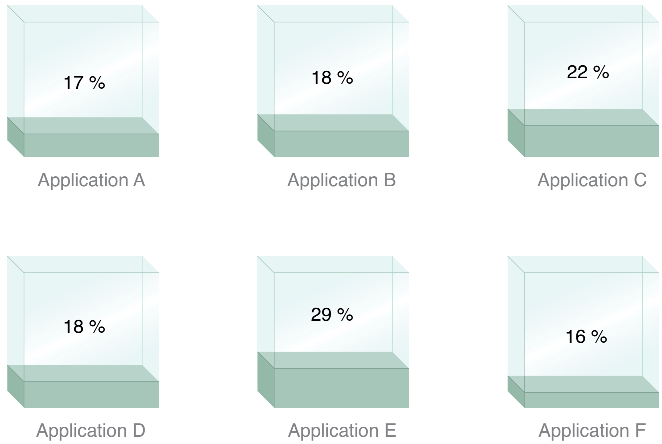


Figure 3.12
Different applications developed independently can result in significant amounts of redundant functionality. The applications displayed were delivered with various levels of solution logic that, in some form, already existed.

It's Not as Efficient as It Appears

Because of the tactical focus on delivering solutions for specific process requirements, the scope of development projects is highly targeted. Therefore, there is the constant perception that business requirements will be fulfilled at the earliest possible time. However, by continually building and rebuilding logic that already exists elsewhere, the process is not as efficient as it could be if the creation of redundant logic could be avoided (Figure 3.13).

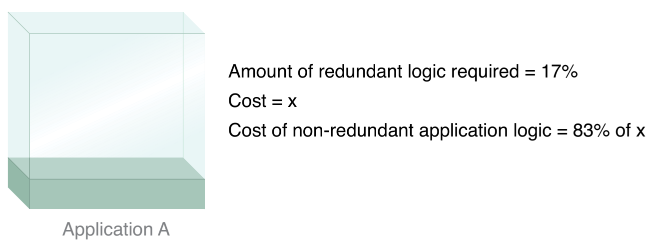


Figure 3.13
Application A was delivered for a specific set of business requirements. Because a subset of these business requirements had already been fulfilled elsewhere, Application A's delivery scope is larger than it has to be.

It Bloats an Enterprise

Each new or extended application adds to the bulk of an IT environment's system inventory (Figure 3.14). The ever-expanding hosting, maintenance, and administration demands can inflate an IT department in budget, resources, and size to the extent that IT becomes a significant drain on the overall organization.

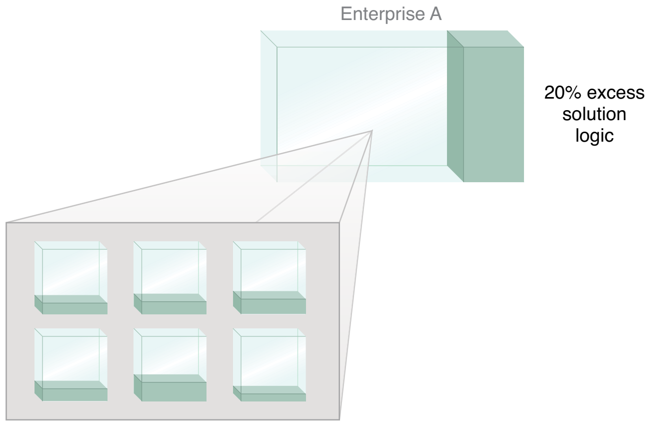


Figure 3.14
This simple diagram portrays an enterprise environment containing applications with redundant functionality. The net effect is a larger enterprise.

It Can Result in Complex Infrastructures and Convoluted Enterprise Architectures

Having to host numerous applications built from different generations of technologies and perhaps even different technology platforms often requires that each will impose unique architectural requirements. The disparity across these "siloed" applications can lead to a counter-federated environment (Figure 3.15), making it challenging to plan the evolution of an enterprise and scale its infrastructure in response to that evolution.

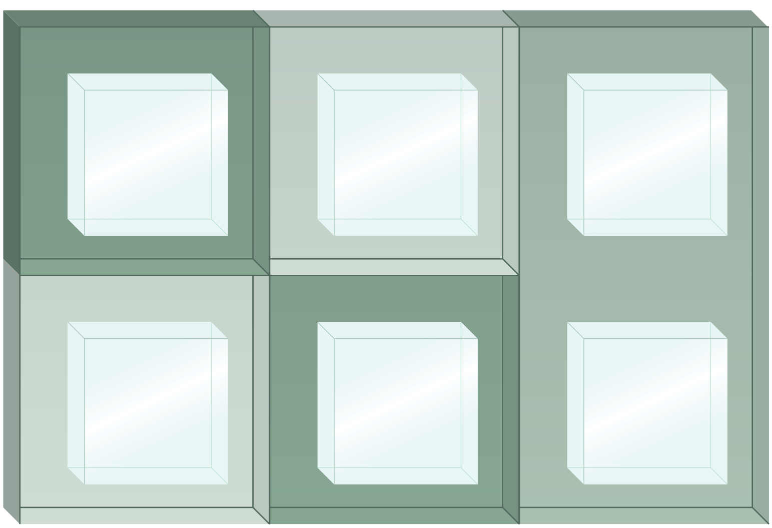


Figure 3.15
Different application environments within the same enterprise can introduce incompatible runtime platforms as indicated by the shaded zones.

Integration Becomes a Constant Challenge

Applications built only with the automation of specific business processes in mind are generally not designed to accommodate other interoperability requirements. Making these types of applications share data at some later point results in a jungle of convoluted integration architectures held together mostly through point-to-point patchwork (Figure 3.16) or requiring the introduction of large middleware layers.

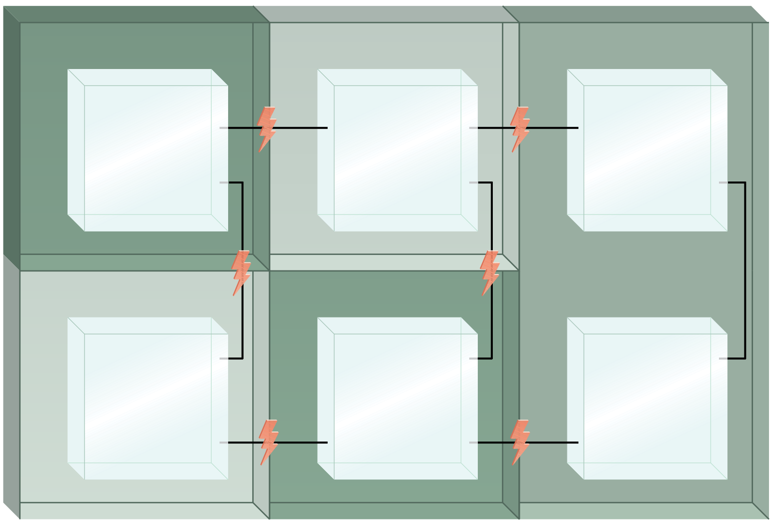


Figure 3.16

A vendor-diverse enterprise can introduce a variety of integration challenges, as expressed by the little lightning bolts that highlight points of concern when trying to bridge proprietary environments.

The Need for Service-Orientation

After repeated generations of traditional distributed solutions, the severity of the previously described problems has been amplified. This is why service-orientation was conceived. It very much represents an evolutionary state in the history of IT in that it combines successful design elements of past approaches with new design elements that leverage conceptual and technology innovation.

The consistent application of the eight design principles we listed earlier results in the widespread proliferation of the corresponding design characteristics:

- increased consistency in how functionality and data is represented
- reduced dependencies between units of solution logic

- reduced awareness of underlying solution logic design and implementation details
- increased opportunities to use a piece of solution logic for multiple purposes
- increased opportunities to combine units of solution logic into different configurations
- increased behavioral predictability
- increased availability and scalability
- increased awareness of available solution logic

When these characteristics exist as real parts of implemented services they establish a common synergy. As a result, the complexion of an enterprise changes as the following distinct qualities are consistently promoted.

Increased Amounts of Reusable Solution Logic

Within a service-oriented solution, units of logic (services) encapsulate functionality not specific to any one application or business process (Figure 3.17). These services are therefore classified as reusable (and agnostic) IT assets.

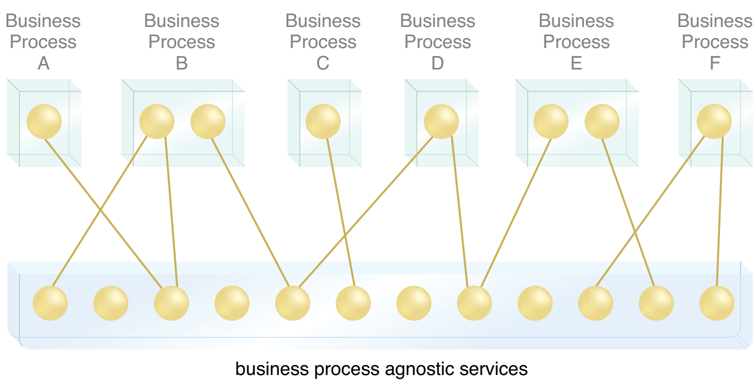


Figure 3.17

Business processes are automated by a series of business process–specific services (top layer) that share a pool of business process–agnostic services (bottom layer). These layers correspond to service models described in Chapter 5.

Reduced Amounts of Application-Specific Logic

Increasing the amount of solution logic not specific to any one application or business process decreases the amount of required application-specific (or "non-agnostic") logic (Figure 3.18). This blurs the lines between standalone application environments by reducing the overall quantity of standalone applications. (See the *Service-Orientation and the Concept of "Application"* section later in this chapter.)

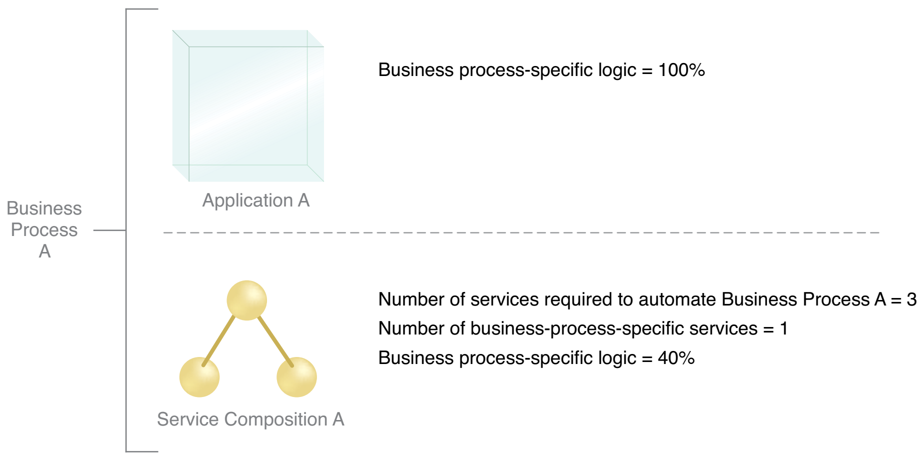


Figure 3.18
Business Process A can be automated by either Application A or Service Composition A. The delivery of Application A can result in a body of solution logic that is all specific to and tailored for the business process. Service Composition A would be designed to automate the process with a combination of reusable services and 40% of additional logic specific to the business process.

Reduced Volume of Logic Overall

The overall quantity of solution logic is reduced because the same solution logic is shared and reused to automate multiple business processes, as shown in Figure 3.19.

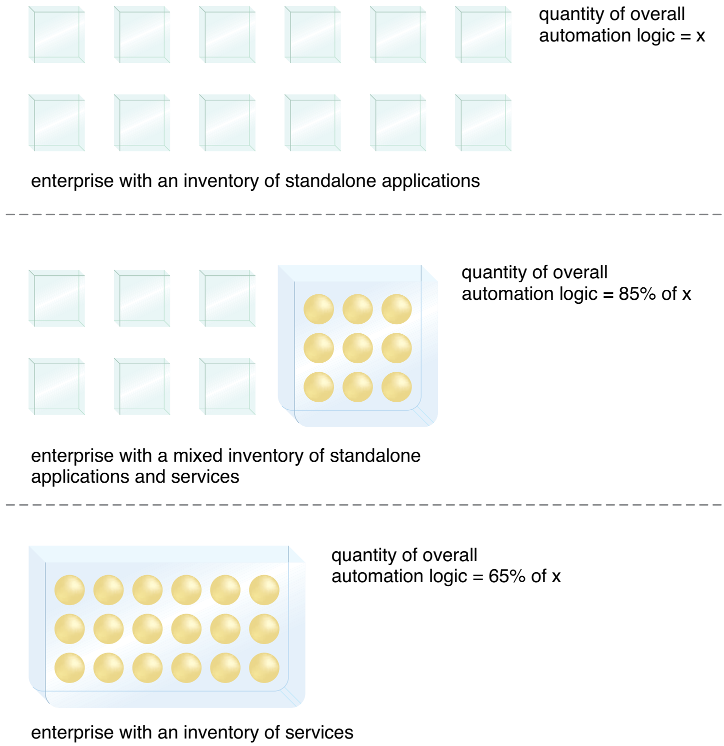


Figure 3.19

The quantity of solution logic shrinks as an enterprise transitions toward a standardized service inventory comprised of "normalized" services. (Service normalization is explained further at the end of Chapter 5.)

Inherent Interoperability

Common design characteristics consistently implemented result in solution logic that is naturally aligned. When this carries over to the standardization of service contracts and their underlying data models, a base level of automatic interoperability is achieved across services, as illustrated in Figure 3.20. (See the *Service-Orientation and the Concept of "Integration"* section later in this chapter.)

NOTE

See Chapter 4 in *SOA Principles of Service Design* for coverage of common challenges introduced by service-orientation.

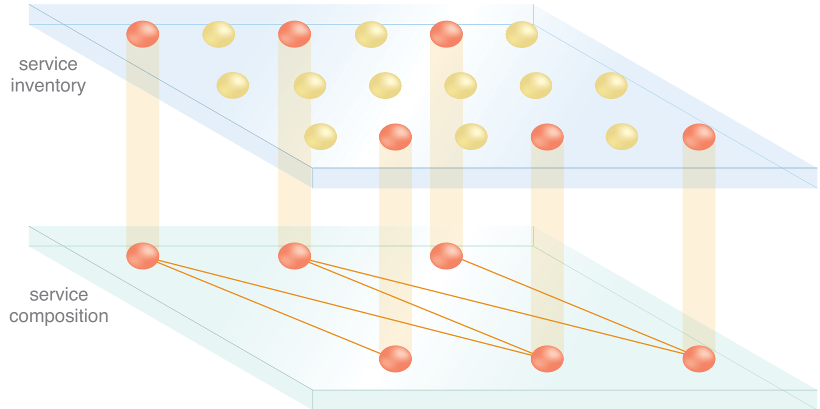


Figure 3.20

Services from different parts of a service inventory can be combined into new compositions. If these services are designed to be intrinsically interoperable, the effort to assemble them into new composition configurations is significantly reduced.

3.3 Effects of Service-Orientation on the Enterprise

There are good reasons to have high expectations from the service-orientation paradigm. But, at the same time, there is much to learn and understand before it can be successfully applied. The following sections explore some of the more common examples.

Service-Orientation and the Concept of "Application"

Having just stated that reuse is not an absolute requirement, it is important to acknowledge the fact that service-orientation does place an unprecedented emphasis on reuse. By establishing a service inventory with a high percentage of reusable and agnostic services, we are now positioning those services as the primary (or only) means by which the solution logic they represent can and should be accessed.

As a result, we make a very deliberate move away from the silos in which applications previously existed. Because we want to share reusable logic whenever possible, we automate existing, new, and augmented business processes through service composition. This results in a shift where more and more business requirements are fulfilled not by building or extending applications, but by simply composing existing services into new composition configurations.

When compositions become more common, the traditional concept of an application or a system or a solution actually begins to fade, along with the silos that contain them. Applications no longer consist of self-contained bodies of programming logic responsible for automating a specific set of tasks (Figure 3.21). What was an application is now just another composition of services, some of which likely participate in other compositions (Figure 3.22).

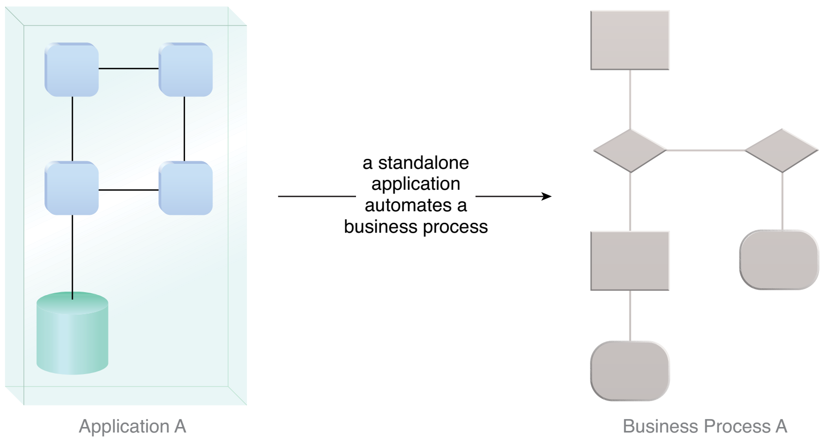


Figure 3.21
The traditional application, delivered to automate specific business process logic.

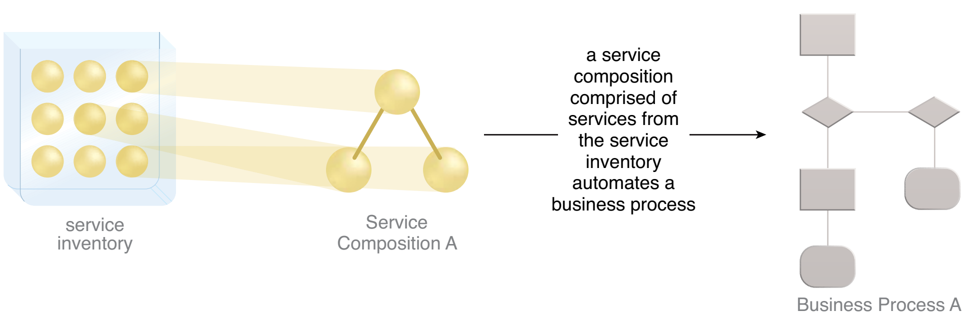


Figure 3.22
The service composition, intended to fulfill the role of the traditional application by leveraging agnostic and non-agnostic services from a service inventory. This essentially establishes a "composite application."

The application therefore loses its individuality. One could argue that a service-oriented application actually does not exist because it is, in fact, just one of many service compositions. However, upon closer reflection, we can see that some of our services (based on the service models established in Chapter 5) are actually not business process agnostic. One service, for example, intentionally represents logic that is dedicated to the automation of just one business task, and therefore not necessarily reusable.

So, single-purpose services can still be associated with the notion of an application. However, within service-oriented computing, the meaning of this term can change to reflect the fact that a potentially large portion of the application logic is no longer exclusive to the application.

Service-Orientation and the Concept of "Integration"

When we revisit the idea of a service inventory consisting of services that have, as per our service-orientation principles, been shaped into standardized and (for the most part) reusable units of solution logic, we can see that this will challenge the traditional perception of "integration."

In the past, integrating something implied connecting two or more applications or programs that may or may not have been compatible (Figure 3.23). Perhaps they were based on different technology platforms or maybe they were never designed to connect with anything outside of their own internal boundary. The increasing need to hook up disparate pieces of software to establish a reliable level of data exchange is what turned integration into an important, high profile part of the IT industry.

Services designed to be "intrinsically interoperable" are built with the full awareness that they will need to interact with a potentially large range of service consumers, most of which will be unknown at the time of their initial delivery. If a significant part of our enterprise solution logic is represented by an inventory of intrinsically interoperable services, it empowers us with the freedom to mix and match these services into infinite composition configurations to fulfill whatever automation requirements come our way.

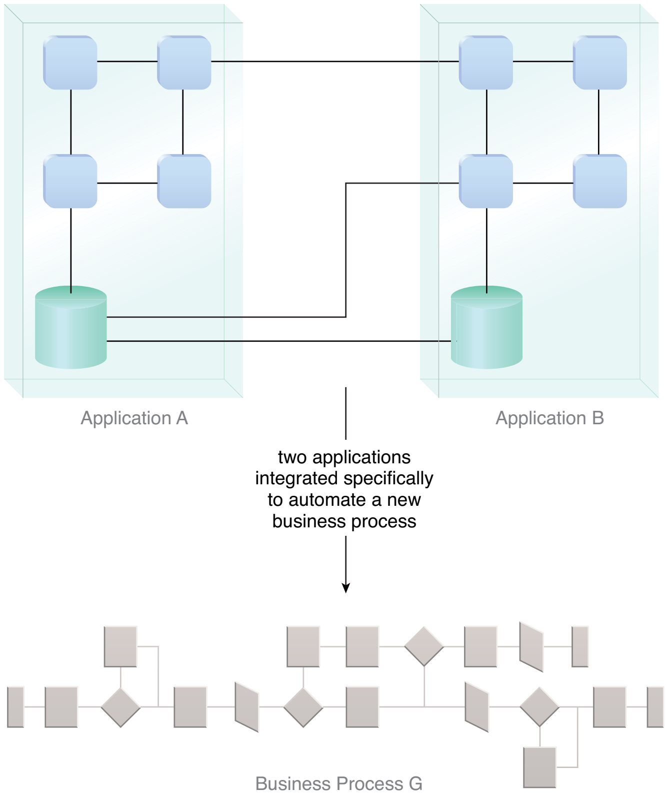


Figure 3.23

The traditional integration architecture, comprised of two or more applications connected in different ways to fulfill a new set of automation requirements (as dictated by the new Business Process G).

As a result, the concept of integration begins to fade. Exchanging data between different units of solution logic becomes a natural and secondary design characteristic (Figure 3.24). Again, though, this is something that can only transpire when a substantial percentage of an organization's solution logic is represented by a quality service inventory. While working toward achieving this environment, there will likely be many requirements for traditional integration between existing legacy systems but also between legacy systems and these services.

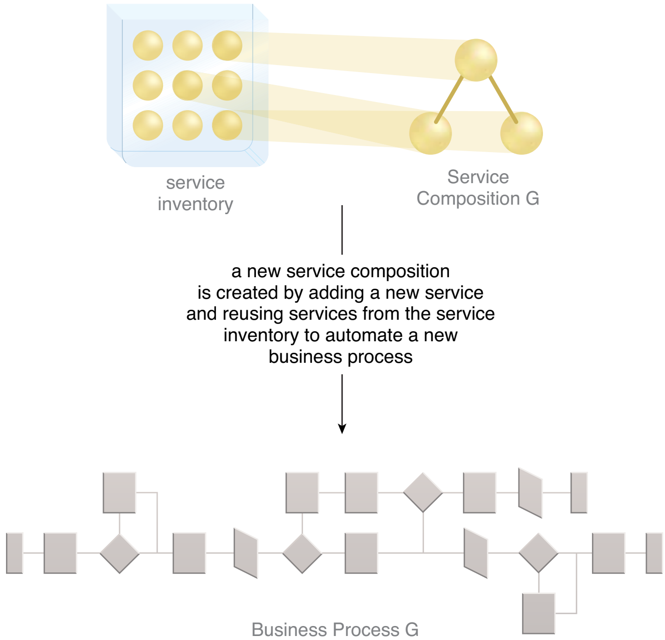
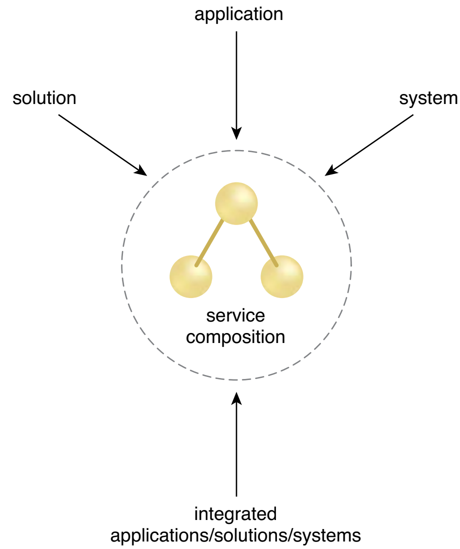


Figure 3.24
A new combination of services is composed together to fulfill the role of traditional integrated applications.

The Service Composition

Applications, integrated applications, solutions, systems—all of these terms and what they have traditionally represented can be directly associated with the service composition (Figure 3.25). As SOA transition initiatives continue to progress within an enterprise, it can be helpful to make a clear distinction between a traditional application (one which may reside alongside an SOA implementation or which may be actually encapsulated by a service) and the service compositions that eventually become more commonplace.

Figure 3.25
A service-oriented solution, application, or system is the equivalent of a service composition.

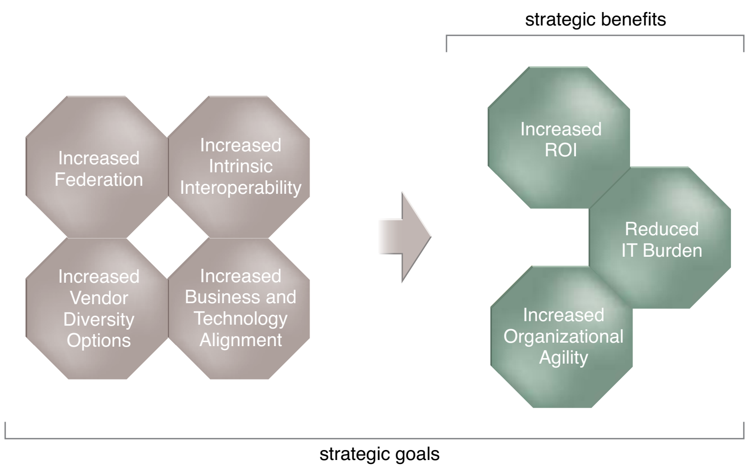


3.4 Goals and Benefits of Service-Oriented Computing

A set of strategic goals and benefits (Figure 3.26) collectively represents the target state we look to achieve when we consistently apply service-orientation to the design of software programs. It is highly beneficial to understand the significance of these goals and benefits because they provide us with constant, overarching context and justification for maintaining our commitment to carrying out service-orientation over the long term.

The upcoming sections describe each of these strategic goals and benefits.

Figure 3.26
The seven identified goals are interrelated and can be further categorized into two groups: strategic goals and resulting benefits. Increased organization agility, increased ROI, and reduced IT burden are concrete benefits resulting from the attainment of the remaining four goals.



Increased Intrinsic Interoperability

Interoperability refers to the sharing of data. The more interoperable software programs are, the easier it is for them to exchange information. Software programs that are not interoperable need to be integrated. Therefore, integration can be seen as a process that enables interoperability. A goal of service-orientation is to establish native interoperability within services to reduce the need for integration (Figure 3.27). As previously explained in the *Effects of Service-Orientation on the Enterprise* section, integration as a concept begins to fade within service-oriented environments.

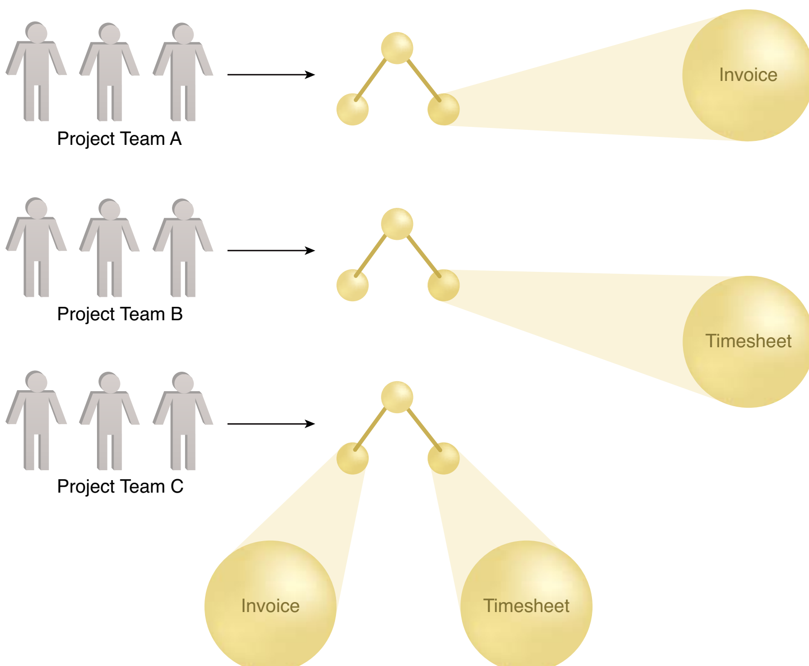


Figure 3.27

Services are designed to be intrinsically interoperable regardless of when and for which purpose they are delivered. In this example, the intrinsic interoperability of the Invoice and Timesheet services delivered by Project Teams A and B allow them to be combined into a new service composition by Project Team C.

Interoperability is specifically fostered through the consistent application of design principles and design standards. This establishes an environment wherein services produced by different projects at different times can be repeatedly assembled together into a variety of composition configurations to help automate a range of business tasks.

Intrinsic interoperability represents a fundamental goal of service-orientation that establishes a foundation for the realization of other strategic goals and benefits. Contract standardization, scalability, behavioral predictability, and reliability are just some of the design characteristics required to facilitate interoperability, all of which are addressed by the service-orientation principles documented in this book.

Each of the eight service-orientation principles supports or contributes to interoperability in some manner. The following are just a few examples:

- *Standardized Service Contract (291)* – Service contracts are standardized to guarantee a baseline measure of interoperability associated with the harmonization of data models.
- *Service Loose Coupling (293)* – Reducing the degree of service coupling fosters interoperability by making individual services less dependent on others and therefore more open for invocation by different service consumers.
- *ServiceA bstraction(294)* – Abstracting details about the service limits all interoperation to the service contract, increasing the long-term consistency of interoperability by allowing underlying service logic to evolve more independently.
- *Service Reusability (295)* – Designing services for reuse implies a high-level of required interoperability between the service and numerous potential service consumers.
- *Service Autonomy (297)* – By raising a service's individual autonomy its behavior becomes more consistently predictable, increasing its reuse potential and thereby its attainable level of interoperability.
- *Service Statelessness (298)* – Through an emphasis on stateless design, the availability and scalability of services increase, allowing them to interoperate more frequently and reliably.
- *Service Discoverability (300)* – Being discoverable simply allows services to be more easily located by those who want to potentially interoperate with them.
- *Service Composability (302)* – Finally, for services to be effectively composable they must be interoperable. The success of fulfilling composability requirements is often tied directly to the extent to which services are standardized and cross-service data exchange is optimized.

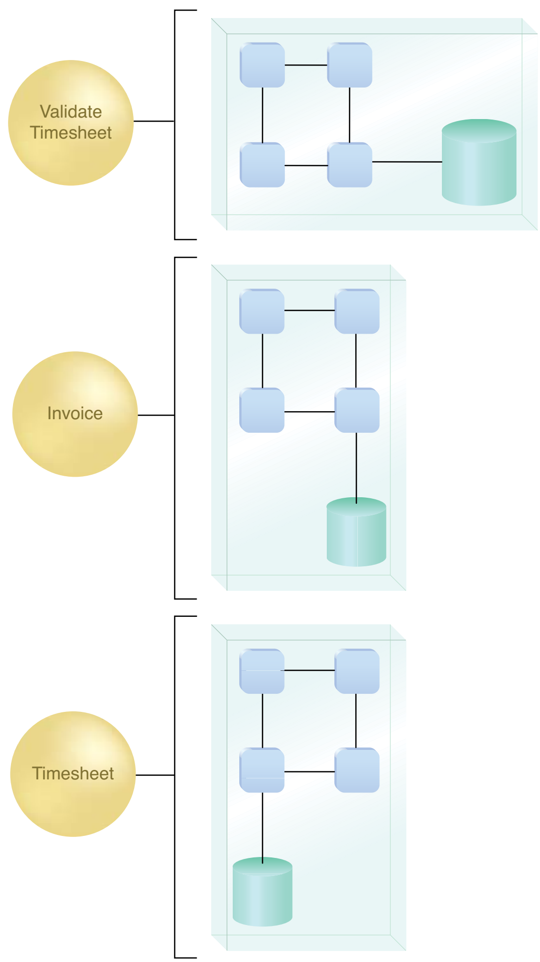
A fundamental goal of applying service-orientation is for interoperability to become a natural by-product, ideally to the extent that a level of intrinsic interoperability is established as a common and expected service design characteristic.

Increased Federation

A federated IT environment is one where resources and applications are united while maintaining their individual autonomy and self-governance. Service-orientation aims to increase a federated perspective of an enterprise to whatever extent it is applied. It accomplishes this through the widespread deployment of standardized and composable services, each of which encapsulates a segment of the enterprise and expresses it in a consistent manner.

In support of increasing federation, standardization becomes part of the extra up-front attention each service receives at design time. Ultimately this leads to an environment where enterprise-wide solution logic becomes naturally harmonized, regardless of the nature of its underlying implementation (Figure 3.28).

Figure 3.28
Three service contracts establishing a federated set of endpoints, each of which encapsulates a different implementation.



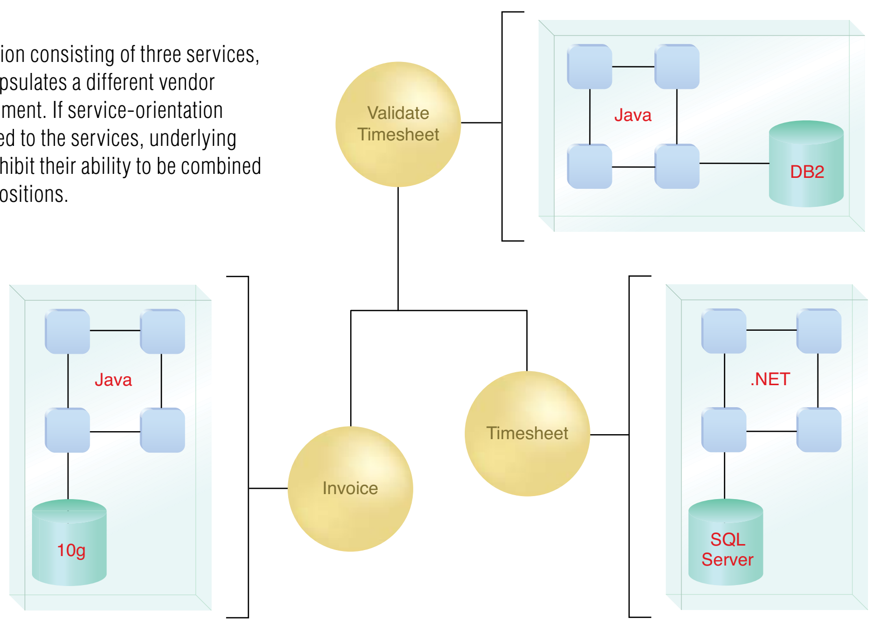
Increased Vendor Diversification Options

Vendor diversification refers to the ability an organization has to pick and choose "best-of-breed" vendor products and technology innovations and use them together within one enterprise. Having a vendor-diverse environment is not necessarily beneficial for an organization; however, having the *option* to diversify when required is beneficial. To have and retain this option requires that its technology architecture not be tied or locked into any one specific vendor platform.

This represents an important state for an enterprise in that it provides the constant freedom for an organization to change, extend, and even replace solution implementations and technology resources without disrupting the overall, federated service architecture. This measure of governance autonomy is attractive because it prolongs the lifespan and increases the financial return of automation solutions.

By designing a service-oriented solution in alignment with but neutral to major vendor SOA platforms and by positioning service contracts as standardized endpoints throughout a federated enterprise, proprietary service implementation details can be abstracted to establish a consistent interservice communications framework. This provides organizations with constant options by allowing them to diversify their enterprise as needed (Figure 3.29).

Figure 3.29
A service composition consisting of three services, each of which encapsulates a different vendor automation environment. If service-orientation is adequately applied to the services, underlying disparity will not inhibit their ability to be combined into effective compositions.



Vendor diversification is further supported by taking advantage of the standards-based, vendor-neutral Web services framework. Because they impose no proprietary communication requirements, services further decrease dependency on vendor platforms. As with any other implementation medium, though, services need to be shaped and standardized through service-orientation to become a federated part of a greater service inventory.

Increased Business and Technology Domain Alignment

The extent to which IT business requirements are fulfilled is often associated with the accuracy with which business logic is expressed and automated by solution logic. Although initial application implementations have traditionally been designed to meet initial requirements, there has historically been a challenge in keeping applications in alignment with business needs as the nature and direction of the business changes.

Service-orientation promotes abstraction on many levels. One of the most effective means by which functional abstraction is applied is the establishment of service layers that accurately encapsulate and represent business models. By doing so, common, pre-existing representations of business logic (business entities, business processes) can exist in implemented form as physical services.

This is accomplished by incorporating a structured analysis and modeling process that requires the hands-on involvement of business subject matter experts in the actual definition of the services (as explained in the *Service-Oriented Analysis (Service Modeling)* section in Chapter 4). The resulting service designs are capable of aligning automation technology with business intelligence on an unprecedented level (Figure 3.30).

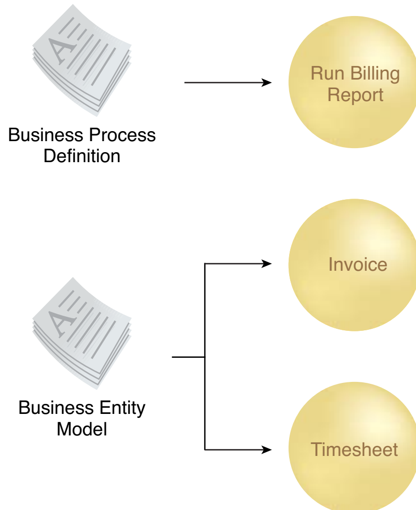
Furthermore, the fact that services are designed to be intrinsically interoperable directly facilitates business change. As business processes are augmented in response to various factors (business climates, new policies, new priorities, etc.) services can be reconfigured into new compositions that reflect the changed business logic. This allows a service-oriented technology architecture to evolve in tandem with the business itself.

Increased ROI

Measuring the return on investment (ROI) of automated solutions is a critical factor in determining just how cost effective a given application or system actually is. The greater the return, the more an organization benefits from the solution. However, the lower the return, the more the cost of automated solutions eats away at an organization's budgets and profits.

Figure 3.30

Services with business-centric functional contexts are carefully modeled to express and encapsulate corresponding business models and logic.



Because the nature of required application logic has increased in complexity and due to ever-growing, non-federated integration architectures that are difficult to maintain and evolve, the average IT department represents a significant amount of an organization's operational budget. For many organizations, the financial overhead required by IT is a primary concern because it often continues to rise without demonstrating any corresponding increase in business value.

Service-orientation advocates the creation of agnostic solution logic—logic that is agnostic to any one purpose and therefore useful for multiple purposes. This multipurpose or reusable logic fully leverages the intrinsically interoperable nature of services. Agnostic services have increased reuse potential that can be realized by allowing them to be repeatedly assembled into different compositions. Any one agnostic service can therefore find itself being repurposed numerous times to automate different business processes as part of different service-oriented solutions.

With this benefit in mind, additional up-front expense and effort is invested into every piece of solution logic to position it as an IT asset for the purpose of repeatable, long-term financial returns. As shown in Figure 3.31, the emphasis on increasing ROI typically goes beyond the returns traditionally sought as part of past reuse initiatives. This has much to do with the fact that service-orientation aims to establish reuse as a common, secondary characteristic within most services.

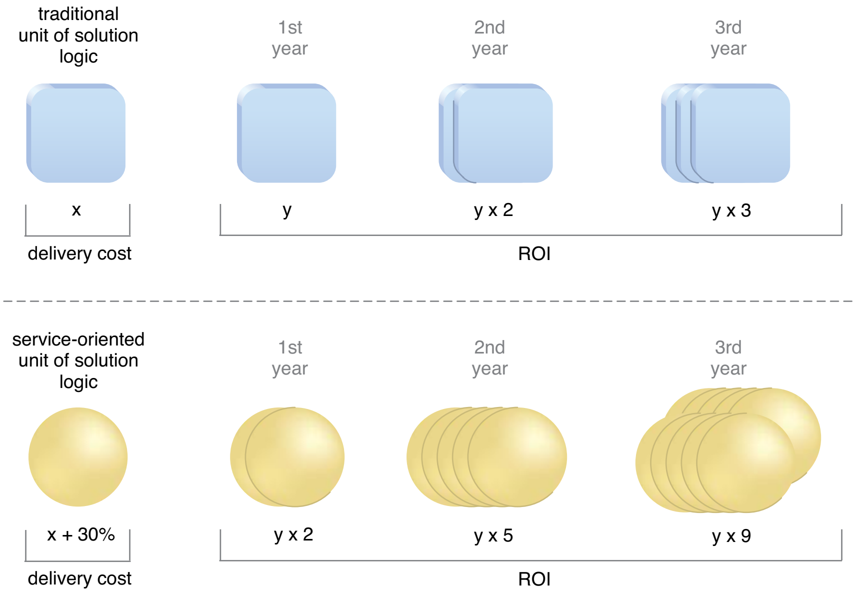


Figure 3.31

An example of the types of formulas being used to calculate ROI for SOA projects. More is invested in the initial delivery with the goal of benefiting from increased subsequent reuse.

It is important to acknowledge that this goal is not simply tied to the benefits traditionally associated with software reuse. Proven commercial product design techniques are incorporated and blended with existing enterprise application delivery approaches to form the basis of a distinct set of service-oriented analysis and design processes (as covered in the chapters in Part II, *Service-Oriented Analysis and Design*).

Increased Organizational Agility

Agility, on an organizational level, refers to efficiency with which an organization can respond to change. Increasing organizational agility is very attractive to corporations, especially those in the private sector. Being able to more quickly adapt to industry changes and outmaneuver competitors has tremendous strategic significance.

An IT department can sometimes be perceived as a bottleneck, hampering desired responsiveness by requiring too much time or resources to fulfill new or changing business requirements. This is one of the reasons agile development methods have gained popularity, as they provide a means of addressing immediate, tactical concerns more rapidly.

Service-orientation is very much geared toward establishing widespread organizational agility. When service-orientation is applied throughout an enterprise, it results in the creation of services that are highly standardized and reusable and therefore agnostic to parent business processes and specific application environments.

As a service inventory is comprised of more and more agnostic services, an increasing percentage of its overall solution logic belongs to no one application environment. Instead, because these services have been positioned as reusable IT assets, they can be repeatedly composed into different configurations. As a result, the time and effort required to automate new or changed business processes is correspondingly reduced because development projects can now be completed with significantly less custom development effort (Figure 3.32).

The net result of this fundamental shift in project delivery is heightened responsiveness and reduced time to market potential, all of which translates into increased organizational agility.

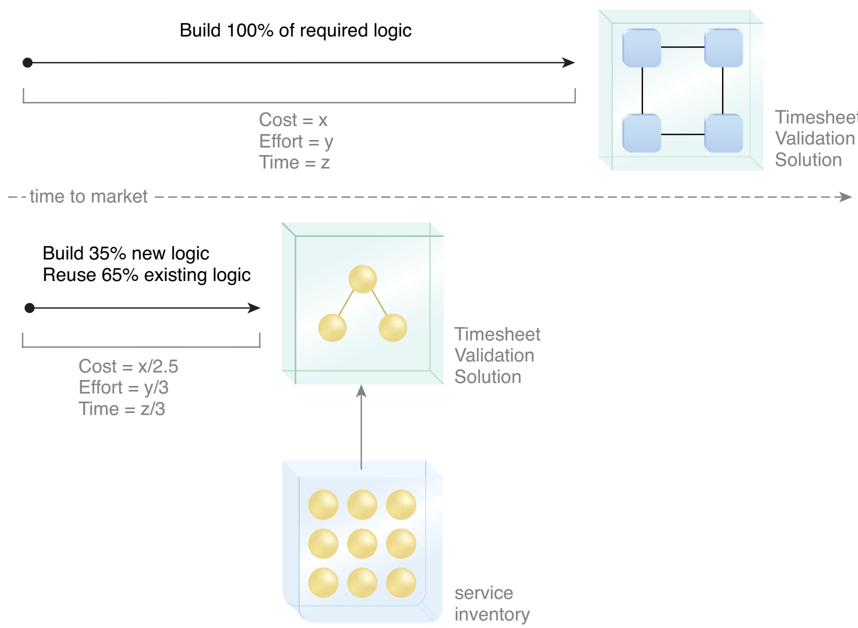


Figure 3.32

The delivery timeline is projected based on the percentage of "net new" solution logic that needs to be built. Though in this example only 35% of new logic is required, the timeline is reduced by around 50% because significant effort is still required to incorporate existing, reusable services from the inventory.

NOTE

Organizational agility represents a target state that organizations work toward as they deliver services and populate service inventories. The organization benefits from increased responsiveness after a significant amount of services is in place. The processes required to model and design those services require more upfront cost and effort than building the corresponding quantity of solution logic using traditional project delivery approaches.

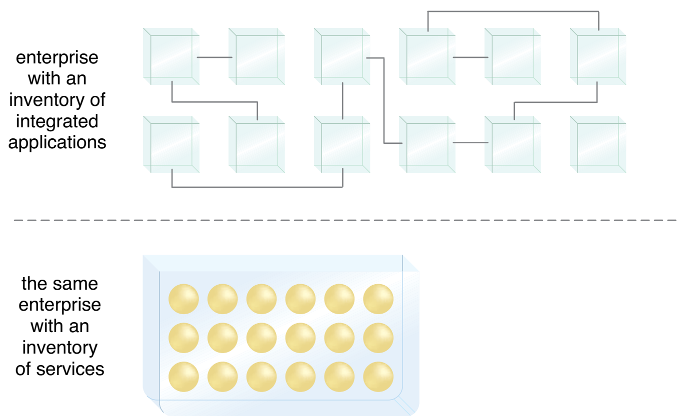
It is therefore important to acknowledge that service-orientation has a strategic focus that intends to establish a highly agile enterprise. This is different from agile development approaches that have more of a tactical focus.

Reduced IT Burden

Consistently applying service-orientation results in an IT enterprise with reduced waste and redundancy, reduced size and operational cost (Figure 3.33), and reduced overhead associated with its governance and evolution. Such an enterprise can benefit an organization through dramatic increases in efficiency and cost-effectiveness.

Figure 3.33

If you were to take a typical automated enterprise and redevelop it entirely with custom, normalized services, its overall size would shrink considerably, resulting in a reduced operational scope.



In essence, the attainment of the previously described goals can create a leaner, more agile IT department, one that is less of a burden on the organization and more of an enabling contributor to its strategic goals.

In summary, the consistent application of service-orientation design principles to individual services that eventually comprise a greater service inventory is the core requirement to achieving the goals and benefits of service-oriented computing (Figure 3.34).

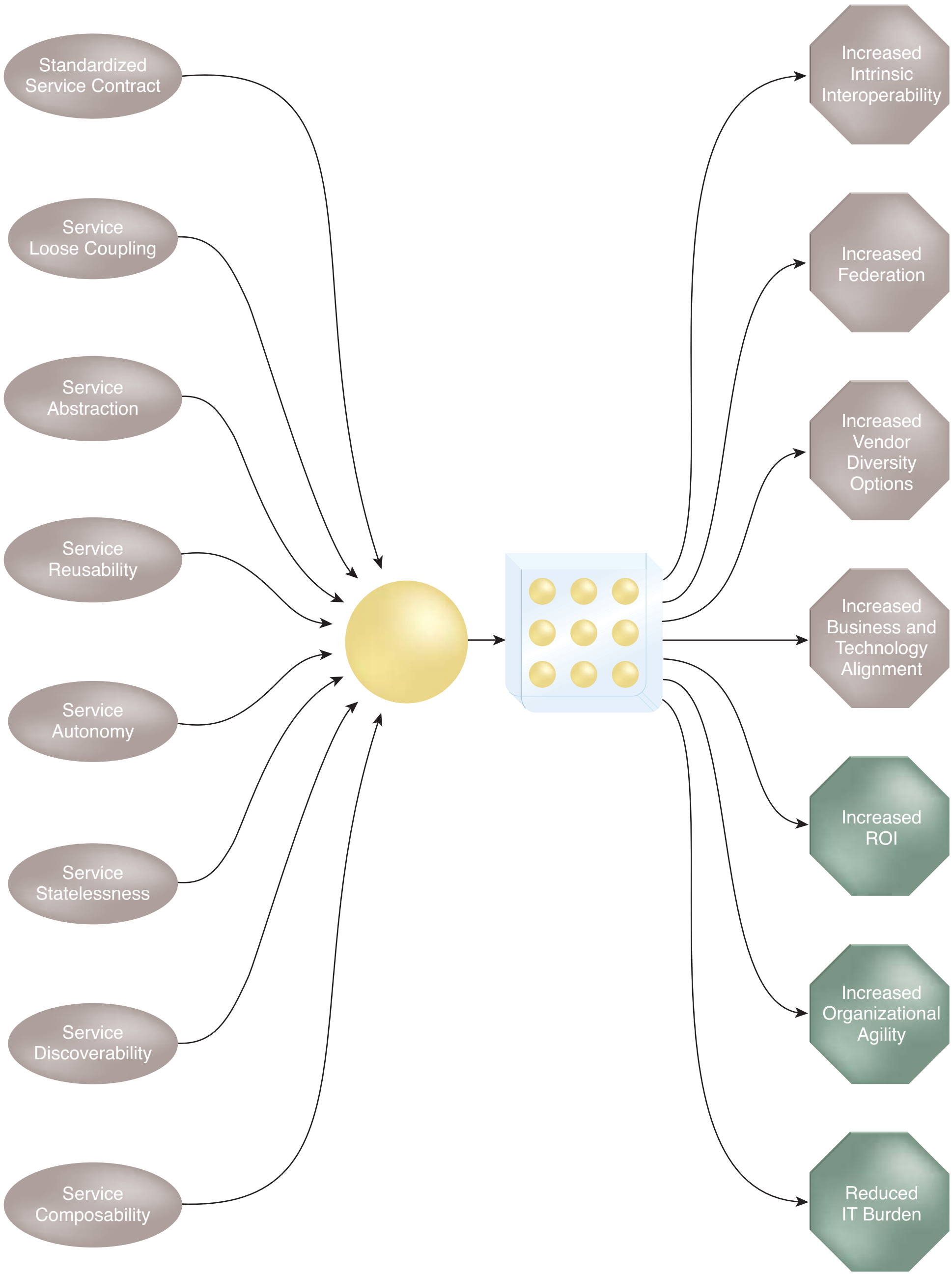


Figure 3.34

The repeated application of service-orientation principles to services that are delivered as part of a collection leads to a target state based on the manifestation of the strategic goals associated with service-oriented computing.

3.5 Four Pillars of Service-Orientation

As previously explained, service-orientation provides us with a well-defined method for shaping software programs into units of service-oriented logic that we can legitimately refer to as services. Each such service that we deliver takes us a step closer to achieving the desired target state represented by the aforementioned strategic goals and benefits.

Proven practices, patterns, principles, and technologies exist in support of service-orientation. However, because of the distinctly strategic nature of the target state that service-orientation aims to establish, there is a set of fundamental critical success factors that act as common prerequisites for its successful adoption. These critical success factors are referred to as *pillars* because they collectively establish a sound and healthy foundation upon which to build, deploy, and govern services.

The four pillars of service-orientation are

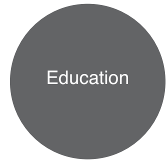
- *Teamwork* – Cross-project teams and cooperation are required.
- *Education* – Team members must communicate and cooperate based on common knowledge and understanding.
- *Discipline* – Team members must apply their common knowledge consistently.
- *Balanced Scope* – The extent to which the required levels of Teamwork, Education, and Discipline need to be realized is represented by a meaningful yet manageable scope.

The existence of these four pillars is considered essential to any SOA initiative. The absence of any one of these pillars to a significant extent introduces a major risk factor. If such an absence is identified in the early planning stages, it can warrant not proceeding with the project until it has been addressed—or the project's scope has been reduced.

Teamwork

Whereas traditional silo-based applications require cooperation among members of individual project teams, the delivery of services and service-oriented solutions requires cooperation across multiple project teams. The scope of the required teamwork is noticeably larger and can introduce new dynamics, new project roles, and the need to forge and maintain new relationships among individuals and departments. Those on the overall SOA team need to trust and rely on each other; otherwise the team will fail.

Education



A key factor to realizing the reliability and trust required by SOA team members is to ensure that they use a common communications framework based on common vocabulary, definitions, concepts, methods, and a common understanding of the target state the team is collectively working to attain. To achieve this common understanding requires common education, not just in general topics pertaining to service-orientation, SOA, and service technologies, but also in specific principles, patterns, and practices, as well as established standards, policies, and methodology specific to the organization.

Combining the pillars of teamwork and education establishes a foundation of knowledge and an understanding of how to use that knowledge among members of the SOA team. The resulting clarity eliminates many of the common risks that have traditionally plagued SOA projects.

Discipline

A critical success factor for any SOA initiative is consistency in how knowledge and practices among a cooperative team are used and applied. To be successful as a whole, team members must therefore be disciplined in how they apply their knowledge and in how they carry out their respective roles. Required measures of discipline are commonly expressed in methodology, modeling, and design standards, as well as governance precepts. Even with the best intentions, an educated and cooperative team will fail without discipline.

Balanced Scope

So far we've established that we need:

- cooperative teams that have…
- a common understanding and education pertaining to industry and enterprise-specific knowledge areas and that…
- we need to consistently cooperate as a team, apply our understanding, and follow a common methodology and standards in a disciplined manner.

In some IT enterprises, especially those with a long history of building silo-based applications, achieving these qualities can be challenging. Cultural, political, and various other forms of organizational issues can arise to make it difficult to attain the necessary organizational changes required by these three pillars. How then can they be realistically achieved? It all comes down to defining a balanced scope of adoption.

The scope of adoption needs to be meaningfully cross-silo, while also realistically manageable. This requires the definition of a balanced scope of adoption of service-orientation.

NOTE

The concept of a balanced scope corresponds directly to the following guideline in the SOA Manifesto:

"The scope of SOA adoption can vary. Keep efforts manageable and within meaningful boundaries."

See Appendix D for the complete SOA Manifesto and the Annotated SOA Manifesto.

Once a balanced scope of adoption has been defined, this scope determines the extent to which the other three pillars need to be established. Conversely, the extent to which you can realize the other three pillars will influence how you determine the scope (Figure 3.35).

Common factors involved in determining a balanced scope include:

- Cultural obstacles
- Authority structures
- Geography
- Business domain alignment
- Available stakeholder support and funding
- Available IT resources

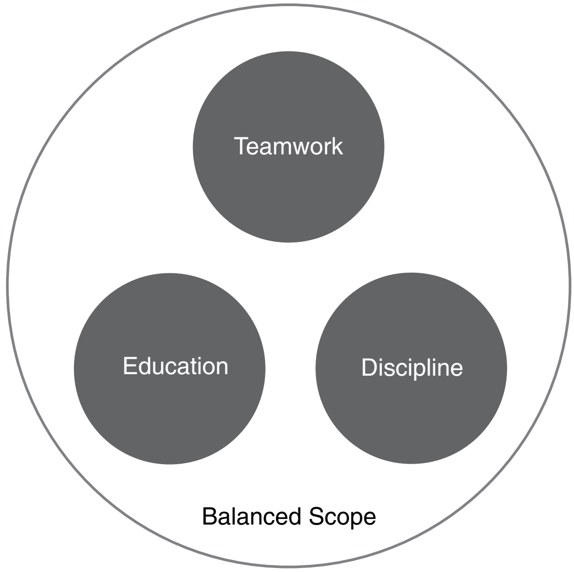


Figure 3.35
The Balanced Scope pillar encompasses and sets the scope at which the other three pillars are applied for a given adoption effort.

A single organization can choose one or more balanced adoption scopes (Figure 3.36). Having multiple scopes results in a domain-based approach to adoption. Each domain establishes a boundary for an inventory of services. Among domains, adoption of service-orientation and the delivery of services can occur independently. This does not result in application silos; it establishes meaningful service domains (also known as "continents of services") within the IT enterprise.

SOA PATTERNS

The domain service inventory originated with the Domain Inventory [338] pattern, which is an alternative to the Enterprise Inventory [340] pattern.

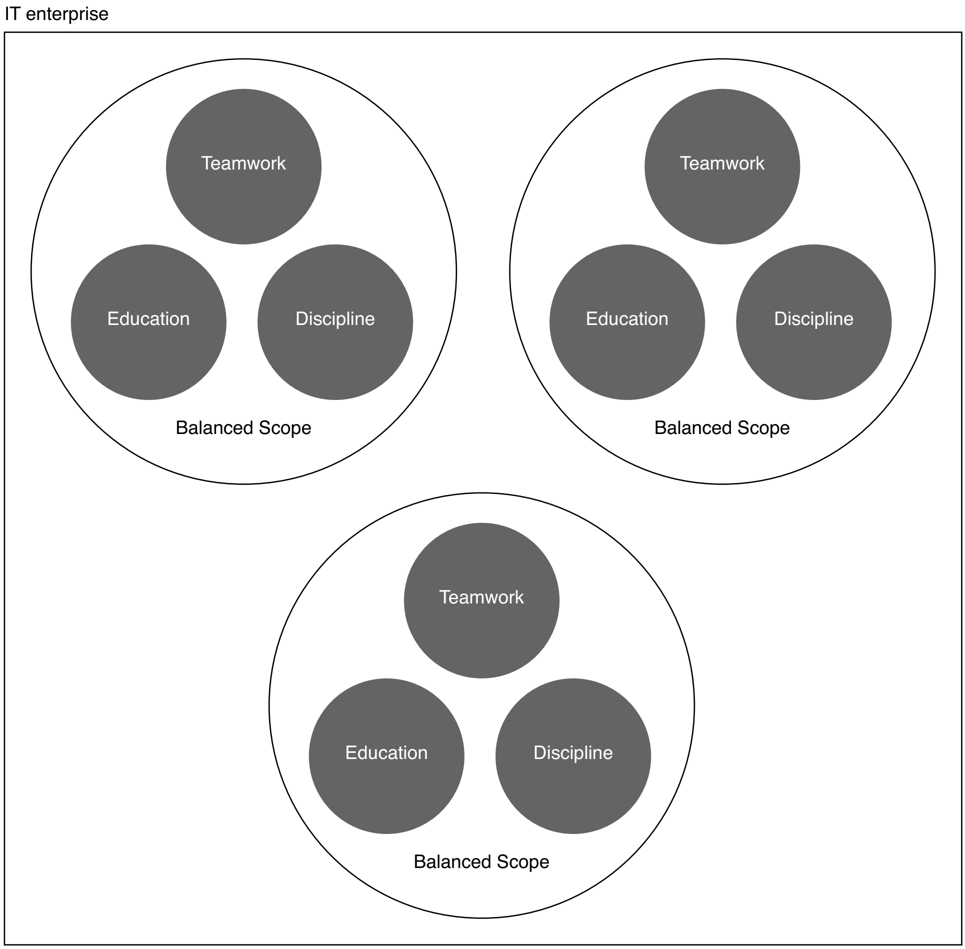


Figure 3.36

Multiple balanced scopes can exist within the same IT enterprise. Each represents a separate domain service inventory that is independently standardized, owned, and governed.

Index

D

I

L

M

N

O

P

V

W