Service-Oriented Architecture
Service-Oriented Architecture

Analysis and Design for Services and Microservices

Thomas Erl

With contributions by Paulo Merson and Roger Stoffers
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

CHAPTER 1: Introduction ......................................................... 1
CHAPTER 2: Case Study Backgrounds ....................................... 13

PART I: FUNDAMENTALS
CHAPTER 3: Understanding Service-Orientation .......................... 19
CHAPTER 4: Understanding SOA ............................................. 59
CHAPTER 5: Understanding Layers with Services and Microservices .... 111

PART II: SERVICE-ORIENTED ANALYSIS AND DESIGN
CHAPTER 6: Analysis and Modeling with Web Services and Microservices .... 139
CHAPTER 7: Analysis and Modeling with REST Services and Microservices .... 159
CHAPTER 8: Service API and Contract Design with Web Services ............... 191
CHAPTER 9: Service API and Contract Design with REST Services and Microservices .......... 219
CHAPTER 10: Service API and Contract Versioning with Web Services and REST Services .................................................. 263

PART III: APPENDICES
APPENDIX A: Service-Orientation Principles Reference ...................... 289
APPENDIX B: REST Constraints Reference .................................... 305
APPENDIX C: SOA Design Patterns Reference .................................. 317
APPENDIX D: The Annotated SOA Manifesto ................................... 367

About the Author ................................................................. 383

Index ...................................................................................... 384
This page intentionally left blank
Acknowledgments ........................................... xix
Reader Services .............................................. xx

CHAPTER 1: Introduction ............................... 1

1.1 How Patterns Are Used in this Book ................. 3
1.2 Series Books That Cover Topics from the First Edition .... 4
1.3 How this Book Is Organized ............................. 6

Part I: Fundamentals ................................. 6
   Chapter 3, Understanding Service-Orientations ............ 6
   Chapter 4, Understanding SOA ........................... 6
   Chapter 5, Understanding Layers with Services and Microservices ... 6

Part II: Service-Oriented Analysis and Design ......... 7
   Chapter 6, Analysis and Modeling with Web Services and Microservices ........ 7
   Chapter 7, Analysis and Modeling with REST Services and Microservices .... 7
   Chapter 8, Service API and Contract Design with Web Services ......... 7
   Chapter 9, Service API and Contract Design with REST Services and Microservices ......... 7
   Chapter 10, Service API and Contract Versioning with Web Services and REST Services .... 7

Part III: Appendices ................................. 7
   Appendix A, Service-Orientations Principles Reference .... 7
   Appendix B, REST Constraints Reference .................. 7
   Appendix C, SOA Design Patterns Reference ............... 8
   Appendix D, The Annotated SOA Manifesto ............... 8

1.4 Page References and Capitalization for Principles,
Constraints, and Patterns ............................... 8
CHAPTER 2: Case Study Backgrounds .......................... 13

2.1 How Case Studies Are Used ............................. 14
2.2 Case Study Background #1: Transit Line Systems, Inc. .. 14
2.3 Case Study Background #2: Midwest University
Association ................................................... 15

PART I: FUNDAMENTALS

CHAPTER 3: Understanding Service-Orientation .............. 19

3.1 Introduction to Service-Orientation ..................... 20
Services in Business Automation .......................... 21
Services Are Collections of Capabilities ................... 22
Service-Orientation as a Design Paradigm ................. 24
Service-Orientation Design Principles ..................... 26

3.2 Problems Solved by Service-Orientation ............... 29
Silo-based Application Architecture ....................... 29
It Can Be Highly Wasteful ............................... 31
It’s Not as Efficient as It Appears ......................... 32
It Bloats an Enterprise ................................. 32
It Can Result in Complex Infrastructures and Convoluted
Enterprise Architectures ................................. 33
Integration Becomes a Constant Challenge ............... 34
The Need for Service-Orientation ......................... 34
Increased Amounts of Reusable Solution Logic .......................... 35
Reduced Amounts of Application-Specific Logic ......................... 36
Reduced Volume of Logic Overall ........................................... 36
Inherent Interoperability ....................................................... 37

3.3 Effects of Service-Orientation on the Enterprise ..................... 38
Service-Orientation and the Concept of “Application” ................. 38
Service-Orientation and the Concept of “Integration” .................. 40
The Service Composition ....................................................... 42

3.4 Goals and Benefits of Service-Oriented Computing ................ 43
Increased Intrinsic Interoperability .......................................... 44
Increased Federation ............................................................ 46
Increased Vendor Diversification Options ................................. 47
Increased Business and Technology Domain Alignment .............. 48
Increased ROI ........................................................................ 48
Increased Organizational Agility .............................................. 50
Reduced IT Burden ................................................................. 52

3.5 Four Pillars of Service-Orientation ...................................... 54
Teamwork ............................................................................. 54
Education ............................................................................. 55
Discipline ............................................................................. 55
Balanced Scope ..................................................................... 55

CHAPTER 4: Understanding SOA .............................................. 59

Introduction to SOA ............................................................... 60

4.1 The Four Characteristics of SOA .......................................... 61
Business-Driven ..................................................................... 61
Vendor-Neutral ...................................................................... 63
Enterprise-Centric ................................................................. 66
Composition-Centric .............................................................. 68
Design Priorities ..................................................................... 69

4.2 The Four Common Types of SOA .......................................... 70
Service Architecture .............................................................. 71
Service Composition Architecture .......................................... 77
Service Inventory Architecture ............................................... 83
Service-Oriented Enterprise Architecture .................................. 85
4.3 The End Result of Service-Orientation and SOA. 86

4.4 SOA Project and Lifecycle Stages 91
   Methodology and Project Delivery Strategies 91
   SOA Project Stages 94
   SOA Adoption Planning 95
   Service Inventory Analysis 96
   Service-Oriented Analysis (Service Modeling) 97
      Step 1: Define Business Automation Requirements 99
      Step 2: Identify Existing Automation Systems 99
      Step 3: Model Candidate Services 100
   Service-Oriented Design (Service Contract) 101
   Service Logic Design 103
   Service Development 103
   Service Testing 103
   Service Deployment and Maintenance 105
   Service Usage and Monitoring 105
   Service Discovery 106
   Service Versioning and Retirement 106
   Project Stages and Organizational Roles 107

CHAPTER 5: Understanding Layers with Services and Microservices 111

5.1 Introduction to Service Layers 113
   Service Models and Service Layers 113
   Service and Service Capability Candidates 115

5.2 Breaking Down the Business Problem 115
   Functional Decomposition 115
   Service Encapsulation 116
   Agnostic Context 117
   Agnostic Capability 119
   Utility Abstraction 120
   Entity Abstraction 121
   Non-Agnostic Context 122
   Micro Task Abstraction and Microservices 123
   Process Abstraction and Task Services 123
PART II: SERVICE-ORIENTED ANALYSIS AND DESIGN

CHAPTER 6: Analysis and Modeling with Web Services and Microservices ........................................... 139

6.1 Web Service Modeling Process ......................... 140
Case Study Example ........................................... 141
Step 1: Decompose the Business Process
(into Granular Actions) ........................................ 142
Case Study Example ........................................... 142
Step 2: Filter Out Unsuitable Actions ...................... 144
Case Study Example ........................................... 145
Step 3: Define Entity Service Candidates ................. 146
Case Study Example ........................................... 146
Step 4: Identify Process-Specific Logic ..................... 149
Case Study Example ........................................... 149
Step 5: Apply Service-Orientation ......................... 150
Step 6: Identify Service Composition Candidates .......... 151
Case Study Example ........................................... 151
Step 7: Analyze Processing Requirements ................. 152
Case Study Example ........................................... 152
Step 8: Define Utility Service Candidates ................. 153
Case Study Example ........................................... 154
Step 9: Define Microservice Candidates .................. 154
Case Study Example ........................................... 155
Step 10: Apply Service-Orientation ......................... 155
Step 11: Revise Service Composition Candidates .......... 156
Case Study Example ........................................... 156
Step 12: Revise Capability Candidate Grouping ........... 157
CHAPTER 7: Analysis and Modeling with REST Services and Microservices

7.1 REST Service Modeling Process

Case Study Example

Step 1: Decompose Business Process (into Granular Actions)

Case Study Example

Step 2: Filter Out Unsuitable Actions

Case Study Example

Step 3: Define Entity Service Candidates

Case Study Example

Step 4: Identify Process-Specific Logic

Case Study Example

Step 5: Identify Resources

Case Study Example

Step 6: Associate Service Capabilities with Resources and Methods

Case Study Example

Step 7: Apply Service-Orientation

Case Study Example

Step 8: Identify Service Composition Candidates

Case Study Example

Step 9: Analyze Processing Requirements

Case Study Example

Step 10: Define Utility Service Candidates (and Associate Resources and Methods)

Case Study Example

Step 11: Define Microservice Candidates (and Associate Resources and Methods)

Case Study Example

Step 12: Apply Service-Orientation

Case Study Example

Step 13: Revise Candidate Service Compositions

Case Study Example

Step 14: Revise Resource Definitions and Capability Candidate Grouping
7.2 Additional Considerations ........................................ 183
   Uniform Contract Modeling and REST Service Inventory
   Modeling .......................................................... 183
   REST Constraints and Uniform Contract Modeling .............. 186
   REST Service Capability Granularity ............................ 188
   Resources vs. Entities ........................................... 189

Chapter 8: Service API and Contract Design with
Web Services ....................................................... 191

8.1 Service Model Design Considerations ....................... 193
   Entity Service Design ........................................ 193
   Utility Service Design ......................................... 194
   Microservice Design ......................................... 196
   Task Service Design ........................................ 196
   Case Study Example .......................................... 198

8.2 Web Service Design Guidelines ............................... 208
   Apply Naming Standards ..................................... 208
   Apply a Suitable Level of Contract API Granularity ........... 210
   Case Study Example .......................................... 212
   Design Web Service Operations to Be Inherently Extensible .... 212
   Case Study Example .......................................... 213
   Consider Using Modular WSDL Documents .................... 214
   Case Study Example .......................................... 214
   Use Namespaces Carefully .................................... 215
   Case Study Example .......................................... 215
   Use the SOAP Document and Literal Attribute Values ........ 216
   Case Study Example .......................................... 217

Chapter 9: Service API and Contract Design with
REST Services and Microservices ............................... 219

9.1 Service Model Design Considerations ....................... 221
   Entity Service Design ........................................ 221
   Utility Service Design ......................................... 222
   Microservice Design ......................................... 223
   Task Service Design ........................................ 225
   Case Study Example .......................................... 226
10.5 Versioning Strategies ............................................ 282
    The Strict Strategy (New Change, New Contract) ........... 282
        Pros and Cons .................................................. 283
    The Flexible Strategy (Backwards Compatibility) ........... 283
        Pros and Cons .................................................. 284
    The Loose Strategy (Backwards and Forwards Compatibility) 284
        Pros and Cons .................................................. 284
    Strategy Summary .................................................. 285

10.6 REST Service Versioning Considerations ................. 286

PART III: APPENDICES

APPENDIX A: Service-Orientation Principles Reference ........ 289
APPENDIX B: REST Constraints Reference ....................... 305
APPENDIX C: SOA Design Patterns Reference ................. 317
    What’s a Design Pattern? ....................................... 318
    What’s a Design Pattern Language? .......................... 320
    Pattern Profiles .................................................. 321

APPENDIX D: The Annotated SOA Manifesto ...................... 367
    The SOA Manifesto ............................................... 368
    The SOA Manifesto Explored .................................... 369
        Preamble ....................................................... 370
        Priorities ..................................................... 371
        Guiding Principles .......................................... 375

About the Author .................................................... 383

Index ................................................................. 384
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
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.
Fundamentals

Chapter 3: Understanding Service-Oriented
Chapter 4: Understanding SOA
Chapter 5: Understanding Layers with Services and Microservices
Chapter 3

Understanding Service-Orientation

3.1 Introduction to Service-Orientation
3.2 Problems Solved by Service-Orientation
3.3 Effects of Service-Orientation on the Enterprise
3.4 Goals and Benefits of Service-Oriented Computing
3.5 Four Pillars of 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](image)

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).

![Figure 3.2](image)

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.
3.1 Introduction to Service-Orientation

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).

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.
Chapter 3: Understanding Service-Orientation

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 3.4
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).

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

“I can:
- drive
- fill out a waybill
- collect payment
  etc.”
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
3.1 Introduction to Service-Orienta...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.

- **Service Abstraction (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.
• **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.
3.2 Problems Solved by Service-Orientation

- **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.

<table>
<thead>
<tr>
<th>SOA PATTERNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

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).
Chapter 3: Understanding Service-Orientation

Valide Timesheet Business Process
Step 1 ...
Step 2 ...
Step 3 ...
...

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

Development cost = x
Yearly operational cost = y
Estimated yearly savings
due to increased productivity = (x/2) - y

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.
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.

Amount of redundant logic required = 17%
Cost = x
Cost of non-redundant application logic = 83% of x

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.
3.2 Problems Solved by Service-Orientation

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

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
3.2 Problems Solved by Service-Orientation

- 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](image)

*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.
3.2 Problems Solved by Service-Orientation

![Diagram showing quantities of overall automation logic for different types of enterprises:]
- Enterprise with an inventory of standalone applications: Quantity of overall automation logic = $x$.
- Enterprise with a mixed inventory of standalone applications and services: Quantity of overall automation logic = 85% of $x$.
- Enterprise with an inventory of services: Quantity of overall automation logic = 65% of $x$.

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

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](image)

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.

- **Service Abstraction** (294) – Abstracting details about the service limits all interoperability 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 life-span 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.
3.4 Goals and Benefits of Service-Oriented Computing

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.
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.
3.4 Goals and Benefits of Service-Oriented Computing

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](image)

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.
3.5 Four Pillars of Service-Orientation

**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
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.
Multiple balanced scopes can exist within the same IT enterprise. Each represents a separate domain service inventory that is independently standardized, owned, and governed.
agents. See service agents
agility (organizational), 50-52
agnostic
business process category, 115
defined, 114
Agnostic Capability design pattern, 133, 322
agnostic capability stage (service layers), 119
Agnostic Context design pattern, 133, 323
agnostic context stage (service layers), 117-118
agnostic logic, 23
Annotated SOA Manifesto, 367-382
application services. See utility services
applications, as service compositions, 38-43
architecture
design patterns and, 70
service architecture, 70-76
service composition architecture, 70, 77-83
service inventory architecture, 70, 83-85
service-oriented enterprise architecture, 70, 85-86
Async complex method, 247, 254-256
Atomic Service Transaction design pattern, 198, 324
attribute values for SOAP messages, 216
automation systems, identifying, 99

B
backwards compatibility, 267-270
flexible versioning strategy, 283-284
loose versioning strategy, 284
balanced scope (service-orientation pillar), 55-58, 97
BDSCP (Big Data Science Certified Professional), 11
benefits of service-orientation, 43
Increased Business and Technology Domain Alignment, 48-49
Increased Federation, 46
Increased Intrinsic Interoperability, 44-45
Increased Organizational Agility, 50-52
Increased ROI, 48-50
Increased Vendor Diversification Options, 47-48
Reduced IT Burden, 52-53
Big Data Science Certified Professional (BDSCP), 11
blueprints. See service inventory blueprints
books
mapped to topics from first edition, 4-6
organization of, 6-8
bottom-up project delivery strategy, 91-92
business community, relationship with IT community, 86-90
business-driven (SOA characteristic), 61-63
business models, technology alignment with, 48-49
business processes
decomposition, 115-124, 142, 164
filtering actions, 144, 165
identifying non-agnostic logic, 149, 169
identifying resources, 170-171
business requirements in service-oriented analysis, 99

C
Cache constraint, 186
profile, 310
Canonical Expression design pattern, 209, 325
Canonical Schema design pattern, 194, 222, 326
Canonical Versioning design pattern, 281, 327
Capability Composition design pattern, 83, 134, 328
capability granularity, 210
Capability Recomposition design pattern, 83, 134, 329
case studies
Midwest University Association (MUA)
analyzing processing requirements, 177-178
applying service-orientation, 174
associating service capability candidates with resources, 173-174
background, 15
business process decomposition, 164
complex methods, 259-262
defining entity service candidates, 167-169
defining microservice candidates, 181
defining utility service candidates, 179-180
design considerations for REST service contracts, 226-230
filtering actions, 165
identifying non-agnostic logic, 169-170
identifying resources, 171-172
identifying service composition candidates, 175-176
REST service modeling, 162-163
revising service composition candidates, 182
Transit Line Systems, Inc. (TLS)
   analyzing processing requirements, 152
   background, 14-15
   business process decomposition, 142-144
   defining entity service candidates, 146-149
   defining microservice candidates, 155
   defining utility service candidates, 154
   design considerations for Web services, 198-208
   filtering actions, 145
   identifying non-agnostic logic, 149-150
   identifying service composition candidates, 151
   modular WSDL documents, 214
   namespaces, 215-216
   revising service composition candidates, 156
   SOAP attribute values, 217
   Web service extensibility, 213
   Web service granularity, 212
   Web service modeling, 141
CCP (Cloud Certified Professional), 10
Client-Server constraint, profile, 307
Cloud Certified Professional (CCP), 10
cloud computing, resources for information, 60
Cloud Computing: Concepts, Technology & Architecture, 60
Cloud Computing Design Patterns, 60
coarse-grained granularity, 211
versioning and, 266-267
Code-on-Demand constraint, profile, 315
compatibility. See also versioning
   REST services considerations, 276-279
   versioning and, 267
   backwards compatibility, 267-270
   compatible changes, 273-275
   forwards compatibility, 271-273
   incompatible changes, 275-276
compatibility guarantee, 280
compatible changes, 273-275
Compensating Service Transaction design pattern, 198, 330
custom methods
   case study, 259-262
   designing, 246-249
   stateful methods, 256-258
   stateless methods, 249-256
composition. See service composition
composition architecture. See service composition architecture
Composition Autonomy design pattern, 224, 331
composition-centricity, 68-69, 124
composition controllers, 78, 123
composition members, 78
Concurrent Contracts design pattern, 193, 195, 212, 221, 223, 332
constraint granularity, 210
   versioning and, 266-267
constraints (REST). See also design constraints
   Cache, 186, 310
   Client-Server, 307
   Code-on-Demand, 315
   Layered System, 187, 313-314
   profile table format, 306
   Stateless, 186, 249, 256-257, 308-309
   Uniform Contract, 183, 187, 311-312
   uniform contract modeling and, 186-187
Containerization design pattern, 333
Content Negotiation design pattern, 244-245, 334
Contemporary SOA. See SOA
Contract Denormalization design pattern, 212, 335
contracts. See service contracts
Cross-Domain Utility Layer design pattern, 195, 336
D
data granularity, 210
decomposition of business processes, 142, 164
decomposition stage (service layers), 115-124
Decoupled Contract design pattern, 193, 337
delivery strategies for SOA projects, 91-92
Delta complex method, 247, 252-254
dependencies, versioning and, 264
deployment stage (SOA projects), 105
design considerations
for REST service contracts
  by service model, 221-225
case study, 226-230
guidelines for, 231-258
for uniform contracts, 231
  HTTP complex method design, 246-249
  HTTP header design, 233-235
  HTTP method design, 231-233
  HTTP response code customization, 240-241
  HTTP response code design, 235-236, 239-240
    media types, 242-244
    schema design, 244-245
  stateful complex methods, 256-258
  stateless complex methods, 249-256
for Web service contracts
  by service model, 193-198
case study, 198-208
guidelines for, 208-216
design constraints, conventions for profiles, 8-9
design paradigms, 24-25
design pattern languages. See pattern languages
design patterns
  advantages of, 318-319
  Agnostic Capability, 133, 322
  Agnostic Context, 133, 323
  architecture and, 70
  Atomic Service Transaction, 198, 324
  Canonical Expression, 209, 325
  Canonical Schema, 194, 222, 326
  Canonical Versioning, 281, 327
  Capability Composition, 83, 134, 328
  Capability Recomposition, 83, 134, 329
  Compensating Service Transaction, 198, 330
  Composition Autonomy, 224, 331
  Concurrent Contracts, 193, 195, 212, 221, 223, 332
  Containerization, 333
  Content Negotiation, 244, 245, 334
  Contract Denormalization, 212, 335
  Cross-Domain Utility Layer, 195, 336
  Decoupled Contract, 193, 337
  defined, 318-319
  Domain Inventory, 57, 83, 97, 187, 195, 338
  Dual Protocols, 155, 193, 195, 212, 221, 223, 339
  Enterprise Inventory, 57, 83, 187, 340
  Entity Abstraction, 133, 341
  Entity Linking, 222, 342
  Event-Driven Messaging, 258, 343
  Functional Decomposition, 133, 344
  Idempotent Capability, 252, 345
  Inventory Endpoint, 86, 346
  Legacy Wrapper, 195, 223, 347
  Logic Centralization, 135, 166, 348
  Microservice Deployment, 349
  Micro Task Abstraction, 134, 350
  Non-Agnostic Context, 133, 351
  Partial State Deferral, 198, 352
  Process Abstraction, 134, 353
  profiles, conventions for, 8-9
  profile table format, 321
  Redundant Implementation, 224, 354
  Reusable Contract, 233, 355
  Schema Centralization, 194, 222, 277, 356
  Service Agent, 76, 357
  Service Data Replication, 224, 358
  Service Encapsulation, 67, 133, 359
  Service Façade, 193, 195, 221, 223, 360
  Service Normalization, 135, 166, 361
  State Messaging, 198, 362
  State Repository, 198, 363
    usage in book, 3-4
  Utility Abstraction, 133, 364
  Validation Abstraction, 214, 245, 365
  Version Identification, 279, 366
design principles, 60-61
  list of, 26, 29
  profiles, conventions for, 8-9
  profile table format, 290
  Service Abstraction, 27, 73, 80, 150, 223, 248
    interoperability, 45
      profile, 294
  Service Autonomy, 27, 73, 150, 174, 194
    interoperability, 45
      profile, 297
  Service Composability, 29, 68, 103, 127, 213
    interoperability, 45
      profile, 302-303
  Service Discoverability, 28, 106
    interoperability, 45
      profile, 300-301
Index

Service Loose Coupling, 26, 150, 223
  interoperability, 45
  profile, 293
Service Reusability, 27, 194, 195, 213
  interoperability, 45
  profile, 295-296
Service Statelessness, 27, 73, 198
  interoperability, 45
  profile, 298-299
Standardized Service Contract, 26, 103, 223-224
  interoperability, 45
  profile, 291-292
design priorities, 69
discipline (service-orientation pillar), 55
discovery stage (SOA projects), 106
document attribute value for SOAP messages, 216
Domain Inventory design pattern, 57, 83, 97, 187, 195, 338
domain service inventory, 25
Dual Protocols design pattern, 155, 193, 195, 212, 221, 223, 339

e
education (service-orientation pillar), 55
terprise-centric (SOA characteristic), 66-67
Enterprise Inventory design pattern, 57, 83, 187, 340
terprise resources, 66
tities, resources versus, 189
Entity Abstraction design pattern, 133, 341
tity abstraction stage (service layers), 121
Entity Linking design pattern, 222, 342
tity service candidates
  associating with resources, 172
t defining, 146, 166
tity services
  defined, 113
t design considerations
    for REST service contracts, 221-222
    for Web services, 193-194
errata, 9, 11
Event-Driven Messaging design pattern, 258, 343
extensibility of Web services, 212-213

f
e Federation, Increased Federation goal/benefit, 46
Fetch complex method, 247, 249-250
figures, symbol legend, 9
fine-grained granularity, 211
  versioning and, 266-267
flexible versioning strategy, 283-285
forwards compatibility, 271-273
  loose versioning strategy, 284
functional decomposition, 116
Functional Decomposition design pattern, 133, 344
functional decomposition stage (service layers), 115

g
goals of service-orientation, 43
  Increased Business and Technology Domain Alignment, 48-49
  Increased Federation, 46
  Increased Intrinsic Interoperability, 44-45
  Increased Organizational Agility, 50-52
  Increased ROI, 48-50
  Increased Vendor Diversification Options, 47-48
  Reduced IT Burden, 52-53
granularity
  constraint granularity, versioning and, 266-267
  REST service modeling, 188
  of Web services, 210-212

h
HTML, compatible changes, 278-279
HTTP headers, design and standardization, 233-235
HTTP media types
  designing, 242-244
  schema design, 244-245
HTTP methods
  complex method design, 246-249
  complex methods case study, 259-262
  design and standardization, 231-233
  stateful complex methods, 256-258
  stateless complex methods, 249-256
HTTP response codes
  customization, 240-241
  design and standardization, 235-236, 239-240
Idempotent Capability design pattern, 252, 345
Increased Intrinsic Interoperability, 44-45
Integration in service-orientation, 40-42
interoperability, 37-38, 44-45
inventory architecture. See service inventory architecture
Inventory Endpoint design pattern, 86, 346
IT community, relationship with business community, 86-90
Layered System constraint, 187
profile, 313-314
Legacy Wrapper design pattern, 195, 223, 347
literal attribute value for SOAP messages, 216
logic centralization, 134
Logic Centralization design pattern, 135, 166, 348
loose versioning strategy, 284-285
maintenance stage (SOA projects), 105
media types
designing, 242-244
schema design, 244-245
uniform contract media types, compatibility, 277-279
messages (SOAP), attribute values, 216
methodology for SOA projects, 91-92
methods (HTTP)
complex method design, 246-249
complex methods case study, 259-262
design and standardization, 231-233
stateful complex methods, 256-258
stateless complex methods, 249-256
microservice candidates, defining, 154, 180
microservice candidate stage (service layers), 123
Microservice Deployment design pattern, 349
microservices
defined, 113
design considerations
for REST service contracts, 223-224
for Web services, 196
service capability composition and, 130-131
Micro Task Abstraction design pattern, 134, 350
micro task abstraction stage (service layers), 123
Midwest University Association case study.
See case studies, Midwest University Association (MUA)
modular WSDL documents, 214
monitoring stage (SOA projects), 105-106
namespaces for WSDL documents, 215
naming standards for Web services, 208-209
Next Generation SOA: A Concise Introduction to Service Technology & Service-Orientation, 3
non-agnostic
business process category, 115
defined, 114
Non-Agnostic Context design pattern, 133, 351
non-agnostic context stage (service layers), 122
non-agnostic logic, 23
identifying, 149, 169
notification service website, 11
open-ended pattern languages, 320
orchestrated task services, 114
organizational agility, Increased Organizational Agility goal/benefit, 50-52
organizational roles, SOA project stages and, 107-109
Partial State Deferral design pattern, 198, 352
pattern languages, 320
patterns. See design patterns
pillars of service-orientation, 54
balanced scope, 55-58, 97
discipline, 55
education, 55
teamwork, 54
Prentice Hall Service Technology Series from Thomas Erl, 2, 4, 6, 290, 306, 321
primitive methods, 247
principles. See design principles
Process Abstraction design pattern, 134, 353
process abstraction stage (service layers), 123-124
profiles, conventions for, 8-9
Index

profile tables. See design patterns; design principles; REST constraints
projects. See SOA projects
PubSub complex method, 257-258

R
recomposition. See service composition
Reduced IT Burden, 52-53
Redundant Implementation design pattern, 224, 354
resources
associating service capability candidates with, 172
entities versus, 189
identifying, 170-171
for information, 9
revising definitions, 182-183
response codes (HTTP)
customization, 240-241
design and standardization, 235-236, 239-240
REST
constraints
Cache, 186, 310
Client-Server, 307
Code-on-Demand, 315
Layered System, 187, 313-314
profile table format, 306
Stateless, 186, 249, 256-257, 308-309
Uniform Contract, 183, 187, 311-312
uniform contract modeling and, 186-187
service inventory modeling, uniform contract modeling and, 183-186
service modeling, 160-161
analyzing processing requirements, 176-177
applying service-orientation, 174, 181
associating service capability candidates with resources, 172
business process decomposition, 164
defining entity service candidates, 166
defining microservice candidates, 180
defining utility service candidates, 178
filtering actions, 165
granularity, 188
identifying non-agnostic logic, 169
identifying resources, 170-171
identifying service composition candidates, 175
process for, 165
resources versus entities, 189
revising service capability candidate groupings, 182-183
revising service composition candidates, 181
REST services, 21
backwards compatibility, 268-270
compatibility considerations, 276-279
forwards compatibility, 271-273
service normalization, 135
versioning, 266, 286
website for information, 10
REST service contracts
benefits of, 220
design considerations
by service model, 221-225
case study, 226-230
guidelines for, 231-236, 239-258
return on investment, Increased ROI goal/benefit, 48, 50
reusability of solution logic, 35
Reusable Contract design pattern, 233, 355
ROI (return on investment), 48, 50

S
Schema Centralization design pattern, 194, 222, 277, 356
schemas, designing for media types, 244-245
separation of concerns, 24-25
Service Abstraction design principle, 27, 73, 80, 150, 223, 248
interoperability, 45
profile, 294
Service Agent design pattern, 76, 357
service agents, 76-77
service architecture, 70-76
Service Autonomy design principle, 27, 73, 150, 174, 194
interoperability, 45
profile, 297
service boundaries, 134
service candidates, 115
service capabilities, 76
service capability candidates
analyzing processing requirements, 152, 176-177
associating with resources, 172
composition and recomposition, 127-133
Service Composability design principle, 29, 68, 103, 127, 213
interoperability, 45
profile, 302-303
service composition
applications as, 38-43
defined, 24, 26, 77
of service capability candidates, 127-133
service-orientation and, 124-127
symbols, 24
service composition architecture, 70, 77-83
service composition candidates
identifying, 151, 175
revising, 156, 181
service consumers, 23
service contracts, 21, 74-75
REST
benefits of, 220
design considerations, 221-230
design guidelines, 231-236, 239-258
Web services
benefits of, 192
design considerations, 193-208
design guidelines, 208-216
Service Data Replication design pattern, 224, 358
service deployment and maintenance, 105
service development, 103
Service Discoverability design principle, 28, 106
interoperability, 45
profile, 300-301
service discovery, 106
Service Encapsulation design pattern, 67, 133, 359
service encapsulation stage (service layers), 116-117
Service Façade design pattern, 193, 195, 221, 223, 360
service granularity, 210
service inventories
defined, 25-26
service boundaries, 134
symbols, 25
uniform contract design considerations, 231
HTTP complex method design, 246-249
HTTP header design, 233-235
HTTP method design, 231-233
HTTP response code customization, 240-241
HTTP response code design, 235-236, 239-240
media types, 242-244
schema design, 244-245
stateful complex methods, 256-258
stateless complex methods, 249-256
service inventory analysis, 96-97
service inventory architecture, 70, 83-85
service inventory blueprint, 84, 96-97
service inventory modeling (REST), uniform contract modeling and, 183-186
service layers
decomposition stage, 115-124
defined, 114
service logic design, 103
Service Loose Coupling design principle, 26, 150, 223
interoperability, 45
profile, 293
service modeling
defined, 100
primitive process steps, 112
REST service modeling, 160-161
analyzing processing requirements, 176-177
applying service-orientation, 174, 181
associating service capability candidates with resources, 172
business process decomposition, 164
defining entity service candidates, 166
defining microservice candidates, 180
defining utility service candidates, 178
filtering actions, 165
granularity, 188
identifying non-agnostic logic, 169
identifying resources, 170-171
identifying service composition candidates, 175
process for, 165
resources versus entities, 189
revising service capability candidate groupings, 182-183
revising service composition candidates, 181
Web services, 140
analyzing processing requirements, 152
applying service-orientation, 150, 155
Index

business process decomposition, 142
problems solved by, 29
architecture complexity, 33
defining entity service candidates, 146
efficiency, lack of, 32
defining microservice candidates, 154
testing, 103-104
defining utility service candidates, 153
testing, 103-104
filtering actions, 144
testing, 103-104
identifying non-agnostic logic, 149
service composition and, 124-127
identifying service composition candidates, 151
result of, 86-90
revising service capability candidate groupings, 157
service composition and, 124-127
revising service composition candidates, 156
result of, 86-90

service models
defined, 113
design considerations
for REST service contracts, 221-225
defining, 26
for Web service contracts, 193-198
design considerations
for REST service contracts, 221-225
list of, 113
design considerations
for Web service contracts, 193-198
service normalization, 134
definition, 26
Service Normalization design pattern, 135, 166, 361
design considerations
for REST service contracts, 221-225
for Web service contracts, 193-198
service normalization, 134
defined, 113
design considerations
for REST service contracts, 221-225
for Web service contracts, 193-198
list of, 113
design considerations
for REST service contracts, 221-225
service-orientation
applications in, 38-43
applying in service modeling, 150, 155, 174, 181
defined, 26
design characteristics of, 34-35
application-specific logic, reducing, 36
testing, 103-104
design characteristics of, 34-35
interoperability, 37-38
service-orientation
applications in, 38-43
testing, 103-104
overall solution logic, reducing, 36-37
reusing solution logic, 35
test test test
test
test
as design paradigm, 24-25
elements of, 26
goals and benefits of, 43
Increased Business and Technology Domain
Alignment, 48-49
test
Increased Federation, 46
Increased Intrinsic Interoperability, 44-45
Increased Organizational Agility, 50-52
Increased ROI, 48-50
Increased Vendor Diversification Options, 47-48
Reduced IT Burden, 52-53
service classes
integration and, 40-42
pilars or, 54
balanced scope, 55-58, 97
discipline, 55
education, 55
teamwork, 54

services
as collections of capabilities, 22-23
defined, 21, 26
discipline, 55
explained, 20-21
test
REST services, 21
symbols for, 21-22
Web services, 21

services contracts, 21
Service Statelessness design principle, 27, 73, 198
interoperability, 45
profile, 295-296
design priorities, 69

SOA (service-oriented architecture)
characteristics of, 61-69
business-driven, 61-63
composition-centric, 68-69
test
test
test
test
test
test
enterprise-centric, 66-67
vendor-neutral, 63-65
test
test
test
test
test
test
test

service-orientation design principles. See design
test
test
test
test
test
test
test
test
test
Stateless constraint, 186, 249, 256-257
profile, 308-309
State Messaging design pattern, 198, 362
State Repository design pattern, 198, 363
Store complex method, 247, 250-251
strict versioning strategy, 282-285
structured pattern languages
advantages of, 320
defined, 320
symbols, 21-22
legend, 9
service composition, 24
service inventory, 25

T
task services
defined, 113
design considerations
for REST service contracts, 225
for Web services, 196-198
task service stage (service layers), 123-124
technology architecture. See architecture
testing stage (SOA projects), 103-104
top-down project delivery strategy, 91-92
Trans complex method, 256
Transit Line Systems, Inc. case study. See case
studies, Transit Line Systems, Inc. (TLS), 14

U
Uniform Contract constraint, 183, 187, 245
profile, 311-312
uniform contract media types, compatibility,
277-279
uniform contract modeling
REST constraints and, 186-187
REST service inventory modeling and, 183-186
uniform contracts, design considerations, 231
HTTP complex method design, 246-249
HTTP header design, 233-235
HTTP method design, 231-233
HTTP response code customization, 240-241
HTTP response code design, 235-236, 239-240
media types, 242-244
schema design, 244-245
stateful complex methods, 256-258
stateless complex methods, 249-256
updates, 9
Utility Abstraction design pattern, 133, 364
utility abstraction stage (service layers), 120
utility service candidates, defining, 153, 178
utility services
defined, 113
design considerations
  for REST service contracts, 222-223
  for Web services, 194, 195
V
Validation Abstraction design pattern, 214, 245, 365
vendor diversification, Increased Vendor
  Diversification Options goal/benefit, 47-48
vendor-neutral (SOA characteristic), 63-65
Version Identification design pattern, 279, 366
version identifiers, 279-281
versioning. See also compatibility
compatibility and, 267
  backwards compatibility, 267-270
  compatible changes, 273-275
  forwards compatibility, 271-273
  incompatible changes, 275-276
constraint granularity and, 266-267
dependencies and, 264
REST services, 266, 286
strategies, 282
comparison of, 285
flexible strategy, 283-284
loose strategy, 284
strict strategy, 282-283
version identifiers, 279-281
Web services, 265-266
versioning stage (SOA projects), 106-107
W
Web Service Contract Design and Versioning
  or SOA, 192, 245
Web service contracts
benefits of, 192
design considerations
case study, 198-208
guidelines for, 208-216
by service model, 193-198
Web services
backwards compatibility, 267-268
extensibility, 212-213
forwards compatibility, 271
granularity, 210-212
naming standards, 208-209
service modeling, 140
  analyzing processing requirements, 152
  applying service-orientation, 150, 155
  business process decomposition, 142
  defining entity service candidates, 146
  defining microservice candidates, 154
  defining utility service candidates, 153
  filtering actions, 144
  identifying non-agnostic logic, 149
  identifying service composition candidates, 151
  revising service capability candidate groupings, 157
  revising service composition candidates, 156
service normalization, 135
versioning, 265-266
websites
  www.arcitura.com/notation, 9
  www.bigdatapatterns.org, 3
  www.bigdatascienceschool.com, 11
  www.cloudpatterns.org, 3, 60
  www.cloudschool.com, 10
  www.serviceorientation.com, 10, 290
  www.servicetechbooks.com, 6, 9, 11, 290, 306, 321
  www.servicetechspecs.com, 10
  www.soamaniesto.com, 8
  www.soapatterns.org, 3, 8, 321
  www.soaschool.com, 10
  www.whatiscloud.com, 60
  www.whatisrest.com, 60
WSDL documents
as modules, 214
namespaces, 215