



"You will find in this book a comprehensive and richly detailed interpretation of what [SOA governance] rules and processes are all about and how they can be concretely implemented."

—Massimo Pezzini, VP and Research Fellow, Gartner

"We are using this book as our reference in both the SOA development and implementation work at the NCI CBIIT, as well as the enterprise architecture definition efforts within HL7."

—Charles N. Mead, MD, MSc., National Cancer Institute and Health Level 7 (HL7) Chair

"Accenture sees this book as a milestone that will support the rationale behind selling and delivering SOA governance projects around the world." —Dr. Matthias Ziegler, Accenture / Dr. Jure Zakothik, Accenture / Thomas M. Michelbach, Accenture

"This is a terrific book that will be heavily used..."

—David S. Rogers, IEEE

SOA Governance

Governing Shared Services On-Premise and in the Cloud

*Co-authored and Edited by Thomas Erl,
World's Top-Selling SOA Author*

*Forewords by
Massimo Pezzini
Roberto Medrano*



SOA School
PRESS

Stephen G. Bennett, Clive Gee, Robert Laird,
Anne Thomas Manes, Robert Schneider,
Leo Shuster, Andre Tost, Chris Venable

With contributions from Benjamin Carlyle, Robert Moores, Filippos Santas



service inventory



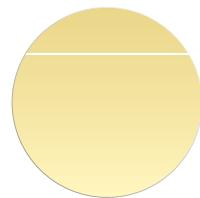
service composition



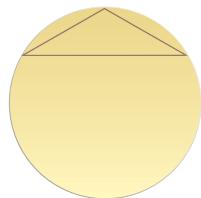
service (labeled)



service layer



service (chorded circle notation)



service accessed
via a uniform interface
(chorded circle notation)



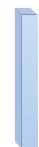
component
or program



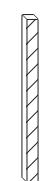
decoupled
service
contract



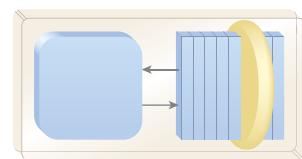
decoupled service
contract accessed
via a uniform
interface



service
agent



firewall



Web service with
service contract



component with
service contract



WSDL
definition



XML Schema
definition



WS-Policy
definition



general machine
processable
document



human-readable
document or content
(including precepts)



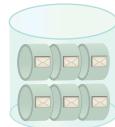
process logic



message



security element
or locked resource



message
queue



repository
or registry



actively
processing



process step or
project/lifecycle stage



physical
server



virtual
server



cloud



zone or
region



conflict
symbol



transition
arrow



human
or role



client
workstation



user
interface



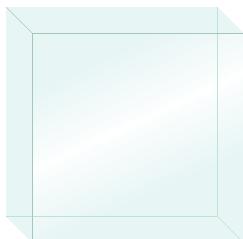
mobile
device



product
or system



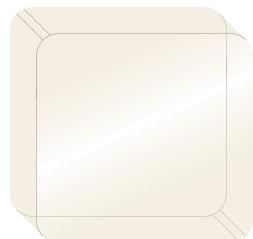
symbols used in conceptual
relationship diagrams



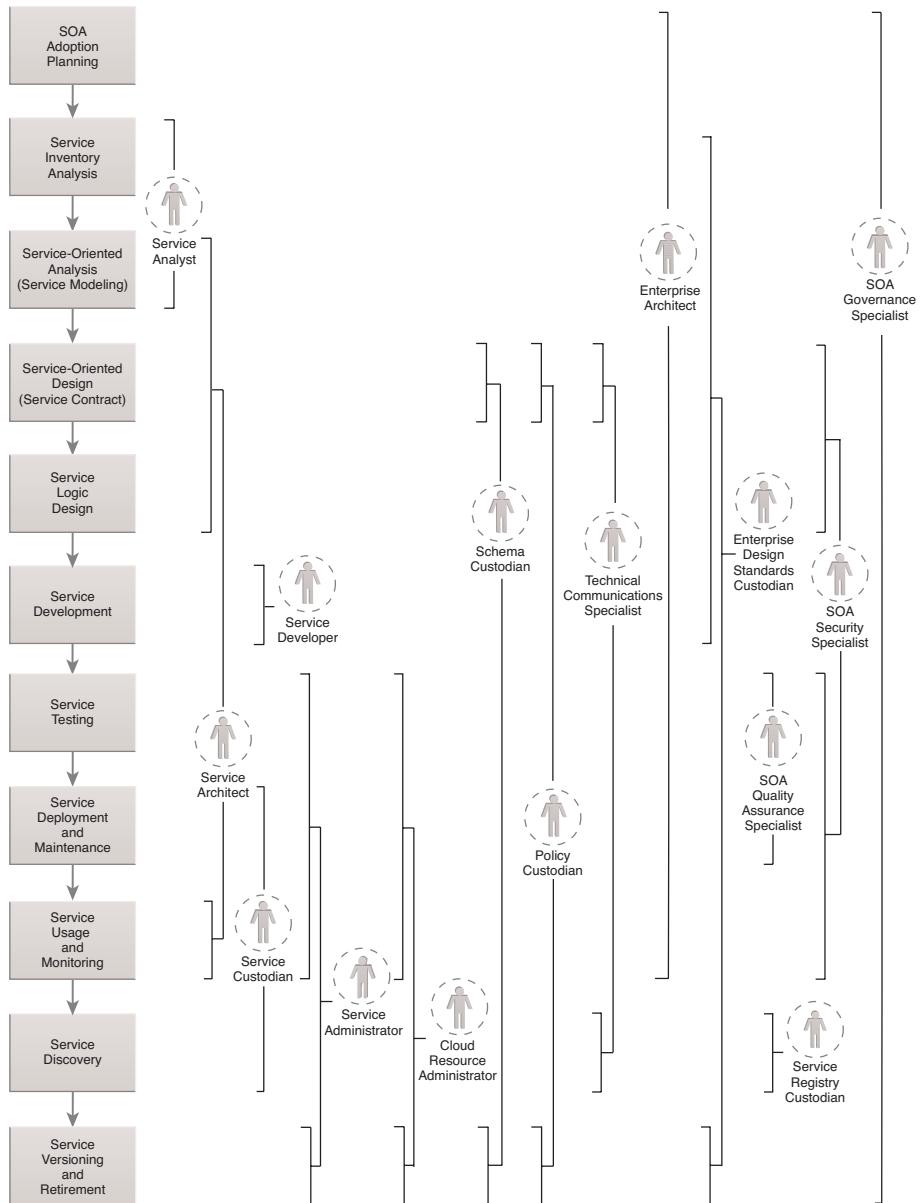
general physical
boundary



service inventory
boundary



service
boundary



Praise for this Book

“Frameworks can be difficult constructs to articulate effectively enough to be useful to practitioners, who are often more interested in answers than guidance. This book on SOA governance provides both thoughtful and carefully crafted narrative, and the supplementation of poignant real-world case studies that will help practitioners calibrate guidance to realities on the ground. This is a terrific book that will be heavily used—with tab stickers, dog-ears, highlighting, and column notes abounding to show for it—as practitioners strategize and subsequently iterate through organizational learnings on their journeys to SOA maturity.”

—*David S. Rogers, Manager, IEEE Conferences Business and Technology Solutions Office, Institute of Electrical and Electronics Engineers (IEEE)*

“This book provides an indispensable guide for establishing a firm SOA governance foundation. Easy to read, comprehensive, pragmatic...excellent job.”

—*Nick Laqua, Enterprise SOA Architect, Cathay Pacific Airways*

“Thomas Erl’s *SOA Governance* book summarizes and clarifies the principles behind this crucial capability for the SOA adoption. Finally, a contribution that serves as a guide for project managers, architects, and any related role that has a common goal: the establishment, administration, and vision behind a service-enabled enterprise. Accenture sees this book as a milestone that will support the rationale behind selling and delivering SOA governance projects around the world. We recommend this book to anyone from technical or business backgrounds interested in service-enabled enterprise to understand why architecture is not only about bits and bytes.”

—*Dr. Matthias Ziegler, Architecture Innovation Lead
Austria, Switzerland, Germany, Accenture,*

—*Dr. Jure Zakotnik, BPM & SOA Project Manager, Technology Architect, Accenture,*

—*Thomas M. Michelbach, Senior Technology Architect, Architecture Innovation, Accenture*

“Achieving your service-oriented goals requires controlled growth and change, which are best accomplished through rigorous governance. The authors of this work drive to the heart of governance and show you how to manage your portfolio of services.”

—*Kevin P. Davis, Ph.D., Software Architect*

“With this book Thomas Erl [and his team] do a great job in outlining a framework to implement an SOA governance program. For each stage of the project lifecycle, necessary governance precepts and processes are described concretely by referring to the service-orientation principles and SOA patterns. This makes it an indispensable source of information for any SOA practitioner or any professional who plans to start an SOA initiative.”

—*Jean-Paul De Baets, Principal SOA Architect, Fedict (Belgian Federal Government Information and Communication Technology Service)*

“Thomas Erl’s *SOA Governance* fills in an important missing piece for any organization wanting to move to—and succeed with—an enterprise commitment to implement SOA and realize its overarching benefits. Of equal importance, however, is the fact that the basic concepts and frameworks that the book instantiates in the context of SOA can also be productively applied in other contexts that are not formally ‘SOA-esque,’ but where complexity is in need of formal governance. For example, we are using this book as our reference in both the SOA development and implementation work at the NCI CBIIT, as well as the enterprise architecture definition efforts within HL7, an international healthcare interoperability Standards Development Organization (SDO) whose purview includes the development of specifications to support computable semantic interoperability using distributed computing paradigms of involving services, messages, and documents.”

—*Charles N. Mead, M.D., MSc., Senior Technical Advisor to the Director, National Cancer Institute Center for Bioinformatics and Information Technology (NCI CBIIT) Chair, Architecture Board, Health Level 7 (HL7)*

“*SOA Governance* is a must-read that provides an in-depth look at the organizational, managerial, procedural, and technical aspects that any SOA project needs to consider. If you’re investing in SOA, you’ll benefit greatly by having this excellent resource available to you as you contend with the many challenges of creating your own SOA governance.”

—*David E. Michalowicz, Principal, Information Systems Engineer, The MITRE Corporation*

“If you are not familiar with SOA governance, this book introduces you to all the relevant stuff needed in a very practical and easy-to-understand manner. Use the processes and precepts shown herein to enable your enterprise [to realize] SOA governance.”

—*Damian Kleer, SOA Architect, DB Systel*

“SOA Governance is the best read on governance and software delivery processes since the publication of RUP; it is the book that defines the standard Service Delivery Processes for all project lifecycle models and defines the necessary conditions and roadmap to reach SOA in the IT organization.”

*—Filippos Santas, IT Architect, Credit Suisse Private Banking,
Switzerland, and Certified SOA Trainer*

“SOA Governance delivers comprehensive coverage of precepts, processes, and roles for every service project lifecycle stage, from analysis through to service retirement. This book truly provides the key to successfully realizing SOA governance within IT projects and the organization as a whole.”

*—Sanjay Singh, Vice President, Rofous Software Pvt. Ltd., Certified SOA Professional,
Certified Scrum Master, IEEE Member*

“SOA Governance blends in nicely with the rest of the Service-Oriented Computing series.... The book gives a good and sensible overview of governance in general as well as SOA governance in particular. After that the book provides more details in the different areas of SOA governance. What I find extra valuable in this book are the case studies since they provide very realistic examples companies can use as a starting point for their own governance work. Together with the tools that are included (such as assessments, templates, and procedures), this book can get your SOA governance program off to a flying start.”

—Herbjörn Wilhelmsen, Enterprise Architect, TUI Nordic

This page intentionally left blank

SOA Governance



The Prentice Hall Service-Oriented Computing Series from Thomas Erl aims to provide the IT industry with a consistent level of unbiased, practical, and comprehensive guidance and instruction in the areas of service-oriented architecture, service-orientation, and the expanding landscape that is shaping the real-world service-oriented computing platform.

For more information, visit www.soabooks.com.

SOA Governance

*Governing Shared Services
On-Premise and in the Cloud*

Co-authored and edited by Thomas Erl

Stephen G. Bennett, Clive Gee, Robert Laird,
Anne Thomas Manes, Robert Schneider,
Leo Shuster, Andre Tost, and Chris Venable

With contributions from Benjamin Carlyle,
Robert Moores, and Filippos Santas



PRENTICE HALL
UPPER SADDLE RIVER, NJ • BOSTON • INDIANAPOLIS • SAN FRANCISCO
NEW YORK • TORONTO • MONTREAL • LONDON • MUNICH • PARIS • MADRID
CAPE TOWN • SYDNEY • TOKYO • SINGAPORE • MEXICO CITY



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informat.com/ph

Library of Congress Cataloging-in-Publication Data

SOA governance : governing shared services on-premise and in the cloud / Thomas Erl ... [et al.].

p. cm.

Includes bibliographical references.

ISBN-13: 978-0-13-815675-6 (hardback : alk. paper)

ISBN-10: (invalid) 0-13-815672-1 (hardback : alk. paper) 1. Service-oriented architecture (Computer science) 2. Business enterprises--Computer networks--Management. I. Erl, Thomas. II. Title: Service-oriented architecture governance.

TK5105.5828.S59 2011

004.6'54--dc22

2011003181

Copyright © 2011 SOA Systems Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671-3447

ISBN-13: 978-0-13-815675-6

ISBN-10: 0-13-815675-1

Text printed in the United States on recycled paper.

First printing April 2011

Editor-in-Chief

Mark L. Taub

Development Editor

Infinet Creative Group

Managing Editor

Kristy Hart

Project Editor

Betsy Harris

Copy Editor

Infinet Creative Group

Senior Indexer

Cheryl Lenser

Proofreaders

Kam Chiu Mok

Shivapriya Nagaraj

Catherine Shaffer

Williams Woods

Publishing

Pamela Janice Yau

Publishing Coordinator

Kim Boedigheimer

Cover Designer

Thomas Erl

Compositors

Bumpy Design

Nonie Ratcliff

Photos

Thomas Erl

Graphics

Infinet Creative Group

*To my supportive wife, Marie, and my children, Sam and Sophia,
for allowing me to take some family time to work on this book.*

—Stephen G. Bennett

*To my family, my colleagues, and my Nespresso® machine,
all of whom supported the creation of this book.*

—Thomas Erl

*Many fine adventures for the past, present, and future for my wife, Amy,
and my sons, Thomas and Jack. You guys make it all worthwhile.*

Thanks to all of the great people that I've worked with on SOA and governance.

I've learned a lot from you.

—Robert Laird

*In memory of my father, Stanley Moores, who died at 45
and whose company and counsel I miss more each year.*

—Robert Moores

To my family in appreciation of their continued support and encouragement.

—Robert D. Schneider

To all my friends and colleagues that helped me make this book a reality.

—Leo Shuster

Thanks and love to my wife, Silke, and my sons, Marc and Jonas.

—Andre Tost

*To my wife, Kelly, and children, Ethan and Sarah, for their loving support;
and to my colleagues for putting up with my endless rants and pontification.*

—Chris Venable

This page intentionally left blank

Contents at a Glance

CHAPTER 1: Introduction	1
CHAPTER 2: Case Study Background	17
PART I: FUNDAMENTALS	
CHAPTER 3: Service-Oriented Computing Fundamentals	23
CHAPTER 4: SOA Planning Fundamentals	49
CHAPTER 5: SOA Project Fundamentals	79
CHAPTER 6: Understanding SOA Governance.....	121
PART II: PROJECT GOVERNANCE	
CHAPTER 7: Governing SOA Projects	153
CHAPTER 8: Governing Service Analysis Stages.....	187
CHAPTER 9: Governing Service Design and Development Stages	221
CHAPTER 10: Governing Service Testing and Deployment Stages	277
CHAPTER 11: Governing Service Usage, Discovery, and Versioning Stages....	315
PART III: STRATEGIC GOVERNANCE	
CHAPTER 12: Service Information and Service Policy Governance.....	369
CHAPTER 13: SOA Governance Vitality	411
CHAPTER 14: SOA Governance Technology	425
PART IV: APPENDICES	
APPENDIX A: Case Study Conclusion.....	453
APPENDIX B: Master Reference Diagrams for Organizational Roles	457
APPENDIX C: Service-Orientation Principles Reference	473
APPENDIX D: SOA Design Patterns Reference	489
APPENDIX E: The Annotated SOA Manifesto	577
APPENDIX F: Versioning Fundamentals for Web Services and REST Services ..	591
APPENDIX G: Mapping Service-Orientation to RUP	617
APPENDIX H: Additional Resources	631
About the Authors	635
About the Contributors.....	641
About the Foreword Contributors.....	643
Index	645

This page intentionally left blank

Contents

Foreword by Massimo Pezzini	xxxi
Foreword by Roberto Medrano	xxxiii
Acknowledgments	xxxv
CHAPTER 1: Introduction	1
1.1 About this Book	3
Who this Book is For.....	3
What this Book Does Not Cover.....	4
<i>This is Not a Book About SOA Management</i>	4
<i>This is Not a Book About Cloud Computing Governance.</i>	4
1.2 Recommended Reading	5
1.3 How this Book is Organized	6
Part I: Fundamentals	6
<i>Chapter 3: Service-Oriented Computing Fundamentals</i>	6
<i>Chapter 4: SOA Planning Fundamentals</i>	6
<i>Chapter 5: SOA Project Fundamentals</i>	6
<i>Chapter 6: Understanding SOA Governance</i>	7
Part II: Project Governance	7
<i>Chapter 7: Governing SOA Projects</i>	7
<i>Chapter 8: Governing Service Analysis Stages</i>	7
<i>Chapter 9: Governing Service Design and Development Stages</i>	8
<i>Chapter 10: Governing Service Testing and Deployment Stages</i>	9
<i>Chapter 11: Governing Service Usage, Discovery, and Versioning Stages</i>	9

Part III: Strategic Governance	10
<i>Chapter 12: Service Information and Service Policy Governance</i>	10
<i>Chapter 13: SOA Governance Vitality</i>	11
<i>Chapter 14: SOA Governance Technology</i>	11
Part IV: Appendices	11
<i>Appendix A: Case Study Conclusion</i>	11
<i>Appendix B: Master Reference Diagrams for Organizational Roles</i>	11
<i>Appendix C: Service-Orientation Principles Reference</i>	11
<i>Appendix D: SOA Design Patterns Reference</i>	11
<i>Appendix E: The Annotated SOA Manifesto</i>	11
<i>Appendix F: Versioning Fundamentals for Web Services and REST Services</i>	12
<i>Appendix G: Mapping Service-Orientation to RUP</i>	12
<i>Appendix H: Additional Resources</i>	12
1.4 Symbols, Figures, and Style Conventions	12
Symbol Legend	12
Mapping Diagrams	12
SOA Principles & Patterns Sections	13
Capitalization	14
1.5 Additional Information	14
Updates, Errata, and Resources (www.soabooks.com)	14
Master Glossary (www.soaglossary.com)	15
Referenced Specifications (www.soaspecs.com)	15
SOASchool.com® SOA Certified Professional (SOACP)	15
CloudSchool.com™ Cloud Certified Professional (CCP)	15
The SOA Magazine (www.soamag.com)	15
Notification Service	16
CHAPTER 2: Case Study Background	17
2.1 How Case Studies are Used	18
2.2 Raysmore Corporation	18
History	18
IT Environment	18
Business Goals and Obstacles	19
2.3 Case Study Continuation	20

PART I: FUNDAMENTALS

CHAPTER 3: Service-Oriented Computing Fundamentals 23

3.1 Basic Terminology	24
Service-Oriented Computing	25
Service-Orientation	26
Service-Oriented Architecture (SOA)	29
Services	31
<i>Services as Components</i>	32
<i>Services as Web Services</i>	32
<i>Services as REST Services</i>	34
SOA Manifesto	34
Cloud Computing	35
IT Resources	35
Cloud	36
On-Premise	37
Cloud Deployment Models	37
Cloud Consumers and Cloud Providers	38
Cloud Delivery Models	38
Service Models	38
<i>Agnostic Logic and Non-Agnostic Logic</i>	39
Service Composition	40
Service Inventory	41
Service Portfolio	41
Service Candidate	42
Service Contract	43
Service-Related Granularity	44
SOA Design Patterns	46
3.2 Further Reading	47

CHAPTER 4: SOA Planning Fundamentals 49

4.1 The Four Pillars of Service-Orientation	51
Teamwork	52
Education	52
Discipline	52
Balanced Scope	53

4.2 Levels of Organizational Maturity	56
Service Neutral Level	57
Service Aware Level	57
Service Capable Level	57
Business Aligned Level	58
Business Driven Level	58
Service Ineffectual Level	58
Service Aggressive Level	59
4.3 SOA Funding Models	60
Platform (Service Inventory) Funding	60
<i>Project Funding Model (Platform)</i>	61
<i>Central Funding Model (Platform)</i>	64
<i>Usage Based Funding Model (Platform)</i>	66
Service Funding	69
<i>Project Funding Model (Service)</i>	69
<i>Central Funding Model (Service)</i>	71
<i>Hybrid Funding Model (Service)</i>	72
<i>Usage Based Funding Model (Service)</i>	74
CHAPTER 5: SOA Project Fundamentals	79
5.1 Project and Lifecycle Stages	81
SOA Adoption Planning	82
Service Inventory Analysis	82
Service-Oriented Analysis (Service Modeling)	84
Service-Oriented Design (Service Contract)	85
Service Logic Design	87
Service Development	87
Service Testing	88
Service Deployment and Maintenance	89
Service Usage and Monitoring	90
Service Discovery	90
Service Versioning and Retirement	91
5.2 Organizational Roles	92
Service Analyst	96
Service Architect	96
Service Developer	97
Service Custodian	98

Cloud Service Owner	98
Service Administrator	100
Cloud Resource Administrator	100
Schema Custodian	102
Policy Custodian	104
Service Registry Custodian	105
Technical Communications Specialist	105
Enterprise Architect	106
Enterprise Design Standards Custodian (and Auditor)	107
SOA Quality Assurance Specialist	109
SOA Security Specialist	110
SOA Governance Specialist	111
Other Roles	112
<i>Educator</i>	112
<i>Business Analyst</i>	113
<i>Data Architect</i>	113
<i>Technology Architect</i>	113
<i>Cloud Technology Professional</i>	114
<i>Cloud Architect</i>	114
<i>Cloud Security Specialist</i>	114
<i>Cloud Governance Specialist</i>	114
<i>IT Manager</i>	115
5.3 Service Profiles	115
Service-Level Profile Structure	117
Capability Profile Structure	118
Additional Considerations	119
<i>Customizing Service Profiles</i>	119
<i>Service Profiles and Service Registries</i>	119
<i>Service Profiles and Service Catalogs</i>	119
<i>Service Profiles and Service Architecture</i>	120
CHAPTER 6: Understanding SOA Governance	121
6.1 Governance 101	122
The Scope of Governance	123
<i>Governance and Methodology</i>	124
<i>Governance and Management</i>	124
<i>Methodology and Management</i>	125
<i>Comparisons</i>	125

The Building Blocks of a Governance System.....	127
<i>Precepts</i>	128
<i>People (Roles)</i>	128
<i>Processes</i>	129
<i>Metrics</i>	129
Governance and SOA	130
6.2 The SOA Governance Program Office (SGPO)	131
6.3 SGPO Jurisdiction Models	133
<i>Centralized Enterprise SGPO</i>	133
<i>Centralized Domain SGPO</i>	134
<i>Federated Domain SGPOs</i>	135
<i>Independent Domain SGPOs</i>	136
6.4 The SOA Governance Program.....	137
Step 1: Assessing the Enterprise (or Domain)	137
<i>Current Governance Practices and Management Styles</i>	138
<i>SOA Initiative Maturity</i>	138
<i>Current Organizational Model</i>	139
<i>Current and Planned Balance of On-Premise and Cloud-based IT Resources</i>	139
Step 2: Planning and Building the SOA Governance Program ..	139
<i>SOA Governance Precepts</i>	139
<i>SOA Governance Processes</i>	141
<i>SOA Governance Roles</i>	143
<i>Additional Components</i>	146
Step 3: Running the SOA Governance Program (Best Practices and Common Pitfalls)	146
<i>Collect the Right Metrics and Have the Right People Use Them</i> ..	146
<i>Provide Transparency and Foster Collaboration</i>	147
<i>Ensure Consistency and Reliability</i>	147
<i>Compliance and Incentives</i>	147
<i>Education and Communication</i>	148
<i>Common Pitfalls</i>	148

PART II: PROJECT GOVERNANCE

CHAPTER 7: Governing SOA Projects	153
7.1 Overview	155
Precepts, Processes, and People (Roles) Sections	156
7.2 General Governance Controls	157
Precepts	157
<i>Service Profile Standards</i>	157
<i>Service Information Precepts</i>	158
<i>Service Policy Precepts</i>	158
<i>Logical Domain Precepts</i>	159
<i>Security Control Precepts</i>	160
<i>SOA Governance Technology Standards</i>	163
Metrics	164
<i>Cost Metrics</i>	164
<i>Standards-related Precept Metrics</i>	165
<i>Threshold Metrics</i>	165
<i>Vitality Metrics</i>	166
Case Study Example	167
7.3 Governing SOA Adoption Planning	169
Precepts	169
<i>Preferred Adoption Scope Definition</i>	169
<i>Organizational Maturity Criteria Definition</i>	171
<i>Standardized Funding Model</i>	172
Processes	173
<i>Organizational Governance Maturity Assessment</i>	173
<i>Adoption Impact Analysis</i>	176
<i>Adoption Risk Assessment</i>	178
People (Roles)	179
<i>Enterprise Architect</i>	179
<i>SOA Governance Specialist</i>	181
Case Study Example	182

CHAPTER 8: Governing Service Analysis Stages 187

8.1 Governing Service Inventory Analysis	192
Precepts	193
<i>Service Inventory Scope Definition</i>	193
Processes	195
<i>Business Requirements Prioritization</i>	195
People (Roles)	197
<i>Service Analyst</i>	197
<i>Enterprise Design Standards Custodian</i>	198
<i>Enterprise Architect</i>	199
<i>SOA Governance Specialist</i>	200
Case Study Example	201
8.2 Governing Service-Oriented Analysis (Service Modeling)	206
Precepts	206
<i>Service and Capability Candidate Naming Standards</i>	206
<i>Service Normalization</i>	207
<i>Service Candidate Versioning Standards</i>	209
Processes	210
<i>Service Candidate Review</i>	210
People (Roles)	212
<i>Service Analyst</i>	212
<i>Service Architect</i>	213
<i>Enterprise Design Standards Custodian</i>	214
<i>Enterprise Architect</i>	215
<i>SOA Governance Specialist</i>	216
Case Study Example	217

**CHAPTER 9: Governing Service Design and
Development Stages 221**

9.1 Governing Service-Oriented Design (Service Contract) ..	223
Precepts	223
<i>Schema Design Standards</i>	223
<i>Service Contract Design Standards</i>	225
<i>Service-Orientation Contract Design Standards</i>	228
<i>SLA Template</i>	229
Processes	231
<i>Service Contract Design Review</i>	231
<i>Service Contract Registration</i>	234

People (Roles)	236
<i>Service Architect</i>	236
<i>Schema Custodian</i>	237
<i>Policy Custodian</i>	238
<i>Technical Communications Specialist</i>	239
<i>Enterprise Design Standards Custodian</i>	241
<i>Enterprise Architect</i>	242
<i>SOA Security Specialist</i>	243
<i>SOA Governance Specialist</i>	245
Case Study Example	246
9.2 Governing Service Logic Design	249
Precepts	249
<i>Service Logic Design Standards</i>	249
<i>Service-Orientation Architecture Design Standards</i>	252
Processes	253
<i>Service Access Control</i>	253
<i>Service Logic Design Review</i>	255
<i>Legal Data Audit</i>	257
People (Roles)	259
<i>Service Architect</i>	259
<i>Enterprise Design Standards Custodian</i>	260
<i>Enterprise Architect</i>	261
<i>SOA Security Specialist</i>	262
<i>SOA Governance Specialist</i>	263
Case Study Example	265
9.3 Governing Service Development	267
Precepts	267
<i>Service Logic Programming Standards</i>	267
<i>Custom Development Technology Standards</i>	268
Processes	270
<i>Service Logic Code Review</i>	270
People (Roles)	272
<i>Service Developer</i>	272
<i>Enterprise Design Standards Custodian</i>	273
<i>Enterprise Architect</i>	274
<i>SOA Governance Specialist</i>	275
Case Study Example	276

CHAPTER 10: Governing Service Testing and Deployment Stages	277
10.1 Governing Service Testing.....	278
Precepts.....	279
<i>Testing Tool Standards</i>	279
<i>Testing Parameter Standards</i>	280
<i>Service Testing Standards</i>	281
<i>Cloud Integration Testing Standards</i>	283
<i>Test Data Usage Guidelines</i>	285
Processes.....	286
<i>Service Test Results Review</i>	286
People (Roles)	287
<i>Service Administrator</i>	287
<i>Cloud Resource Administrator</i>	288
<i>Enterprise Architect</i>	289
<i>SOA Quality Assurance Specialist</i>	290
<i>SOA Security Specialist</i>	291
<i>SOA Governance Specialist</i>	292
Case Study Example	294
10.2 Governing Service Deployment and Maintenance	298
Precepts.....	298
<i>Production Deployment and Maintenance Standards</i>	298
Processes.....	301
<i>Service Certification Review</i>	301
<i>Service Maintenance Review</i>	303
People (Roles)	304
<i>Service Administrator</i>	304
<i>Cloud Resource Administrator</i>	305
<i>Service Custodian</i>	307
<i>Enterprise Architect</i>	308
<i>SOA Quality Assurance Specialist</i>	309
<i>SOA Security Specialist</i>	310
<i>SOA Governance Specialist</i>	311
Case Study Example	312

**Chapter 11: Governing Service Usage, Discovery,
and Versioning Stages 315**

11.1 Governing Service Usage and Monitoring	317
Precepts	317
<i>Runtime Service Usage Thresholds</i>	317
<i>Service Vitality Triggers</i>	320
Processes	323
<i>Service Vitality Review</i>	323
People (Roles)	325
<i>Enterprise Architect</i>	325
<i>Service Architect</i>	326
<i>Service Administrator</i>	327
<i>Cloud Resource Administrator</i>	328
<i>Service Custodian</i>	329
<i>SOA Security Specialist</i>	331
<i>SOA Governance Specialist</i>	332
Case Study Example	333
11.2 Governing Service Discovery	335
Precepts	335
<i>Centralized Service Registry</i>	335
Processes	337
<i>Service Registry Access Control</i>	337
<i>Service Registry Record Review</i>	339
<i>Service Discovery</i>	340
<i>Shared Service Usage Request</i>	342
<i>Shared Service Modification Request</i>	343
People (Roles)	345
<i>Service Custodian</i>	345
<i>Service Registry Custodian</i>	346
<i>Technical Communications Specialist</i>	348
<i>SOA Governance Specialist</i>	348
Case Study Example	350
11.3 Governing Service Versioning and Retirement	352
Precepts	352
<i>Service Versioning Strategy</i>	352
<i>SLA Versioning Rules</i>	354
<i>Service Retirement Notification</i>	356

Processes.....	357
<i>Service Versioning</i>	357
<i>Service Retirement</i>	359
People (Roles)	360
<i>Enterprise Design Standards Custodian</i>	360
<i>Service Administrator</i>	362
<i>Cloud Resource Administrator</i>	363
<i>Schema Custodian</i>	364
<i>Policy Custodian</i>	364
<i>SOA Governance Specialist</i>	365

PART III: STRATEGIC GOVERNANCE

CHAPTER 12: Service Information and Service Policy Governance 369

12.1 Overview	371
Service Data vs. Service Information.....	371
Policies 101	373
12.2 Governance Controls	375
Precepts	375
<i>Enterprise Business Dictionary/Domain Business Dictionary</i>	375
<i>Service Metadata Standards</i>	377
<i>Enterprise Ontology/Domain Ontology</i>	380
<i>Business Policy Standards</i>	382
<i>Operational Policy Standards</i>	384
<i>Policy Centralization</i>	386
Processes.....	389
<i>Data Quality Review</i>	389
<i>Communications Quality Review</i>	391
<i>Information Alignment Audit</i>	393
<i>Policy Conflict Audit</i>	395
People (Roles)	397
<i>Business Analyst</i>	397
<i>Data Architect</i>	399
<i>Schema Custodian</i>	399
<i>Policy Custodian</i>	401
<i>Service Registry Custodian</i>	402

<i>Technical Communications Specialist</i>	403
<i>SOA Quality Assurance Specialist</i>	405
<i>SOA Governance Specialist</i>	406
12.3 Guidelines for Establishing Enterprise Business Models	408
Establish a Service Information Governance Council	408
Assign Business Information Custodians	408
Assign Value to Business Information	409
Relate Service Information Governance to Master Data Management	409
CHAPTER 13: SOA Governance Vitality	411
13.1 Vitality Fundamentals	412
13.2 Vitality Triggers	414
<i>Business vs. Technology Changes</i>	415
<i>Types of Vitality Triggers</i>	416
Strategic Adjustments	416
<i>Strategic Business Adjustment</i>	416
<i>Strategic IT Adjustment</i>	417
Industry Shifts	417
<i>Business Shift</i>	417
<i>Technology Shift</i>	418
Metrics	418
<i>Performance Metrics</i>	419
<i>Compliance Metrics</i>	419
Organizational Shifts	419
Periodic	420
<i>Milestone</i>	420
<i>Time</i>	420
13.3 SOA Governance Vitality Process	421
Identify Activity	421
Assess Activity	422
Refresh Activity	422
Approve Activity	423
Communicate Activity	423

CHAPTER 14: SOA Governance Technology	425
14.1 Understanding SOA Governance Technology	426
SOA Governance Task Types	427
<i>Manual Governance</i>	427
<i>Automated Governance</i>	427
<i>Design-time Governance</i>	428
<i>Runtime Governance</i>	428
<i>On-Premise Governance</i>	428
<i>Cloud Governance</i>	428
<i>Passive Governance</i>	428
<i>Active Governance</i>	429
SOA Governance Technology Types	429
<i>Administrative</i>	429
<i>Monitoring</i>	429
<i>Reporting</i>	430
<i>Enforcement</i>	430
14.2 Common SOA Governance Technology Products	431
Service Registries	431
<i>Task Types</i>	432
<i>Technology Types</i>	432
<i>SOA Project Stages</i>	433
Repositories	433
<i>Task Types</i>	434
<i>Technology Types</i>	434
<i>SOA Project Stages</i>	435
Service Agents	435
<i>Task Types</i>	436
<i>Technology Types</i>	437
<i>SOA Project Stages</i>	437
Policy Systems	437
<i>Task Types</i>	438
<i>Technology Types</i>	438
<i>SOA Project Stages</i>	439
Quality Assurance Tools	439
<i>Task Types</i>	440
<i>Technology Types</i>	440
<i>SOA Project Stages</i>	441
SOA Management Suites	441

Other Tools and Products	442
<i>Technical Editors and Graphic Tools</i>	442
<i>Content Sharing and Publishing Tools</i>	442
<i>Configuration Management Tools</i>	443
<i>Custom SOA Governance Solutions</i>	443
14.3 Guidelines for Acquiring SOA Governance Technology	444
Acquisition Strategies	444
<i>Single Vendor</i>	444
<i>Multiple Vendors</i>	445
<i>Open Source</i>	446
<i>Leased from Cloud Vendor</i>	447
Best Practices	448
<i>Establish Criteria Based on Your Specific Requirements</i>	448
<i>Investigate Customizability</i>	448
<i>Investigate APIs</i>	448
<i>Understand Both Initial and Long-Term Costs</i>	448
<i>Understand Actual Governance Support</i>	449
<i>Take the Time to Create a Quality RFP</i>	449

PART IV: APPENDICES

APPENDIX A: Case Study Conclusion	453
--	------------

APPENDIX B: Master Reference Diagrams for Organizational Roles	457
---	------------

Service Analyst	458
Service Architect	459
Service Developer	460
Service Custodian	460
Service Administrator	461
Cloud Resource Administrator	462
Schema Custodian	463
Policy Custodian	464
Service Registry Custodian	465
Technical Communications Specialist	466
Enterprise Architect	467

Enterprise Design Standards Custodian (and Auditor)	468
SOA Quality Assurance Specialist	469
SOA Security Specialist	470
SOA Governance Specialist (precepts)	471
SOA Governance Specialist (processes)	472
APPENDIX C: Service-Orientation Principles Reference	473
APPENDIX D: SOA Design Patterns Reference	489
APPENDIX E: The Annotated SOA Manifesto	577
APPENDIX F: Versioning Fundamentals for Web Services and REST Services	591
F.1 Versioning Basics	593
Versioning Web Services	593
Versioning REST Services	594
Fine and Coarse-Grained Constraints	595
F.2 Versioning and Compatibility	596
Backwards Compatibility	596
<i>Backwards Compatibility in Web Services</i>	596
<i>Backwards Compatibility in REST Services</i>	597
Forwards Compatibility	599
Compatible Changes	602
Incompatible Changes	604
F.3 REST Service Compatibility Considerations	605
F.4 Version Identifiers	608
F.5 Versioning Strategies	611
The Strict Strategy (New Change, New Contract)	611
<i>Pros and Cons</i>	612
The Flexible Strategy (Backwards Compatibility)	612
<i>Pros and Cons</i>	613

The Loose Strategy (Backwards and Forwards Compatibility)	613
<i>Pros and Cons</i>	614
Summary Table	614
F.6 REST Service Versioning Considerations	615

APPENDIX G: Mapping Service-Orientation to RUP 617

Compatibility of RUP and SOA	618
Overview of RUP (and MSOAM)	619
The Pillars of Service-Orientation and the RUP Principles.....	620
Breadth and Depth Roles and Role Mapping	623
Enterprise and Governance Roles	624
Mapping Service Delivery Project Stages to Disciplines	625
Mapping MSOAM Analysis and Design Stages to RUP Disciplines	626
Service-Orientation and RUP: Gaps	628
Related Reading.....	628
Bibliography.....	629

APPENDIX H: Additional Resources..... 631

About the Authors	635
Stephen G. Bennett	635
Thomas Erl	635
Clive Gee, Ph.D.....	636
Robert Laird.....	637
Anne Thomas Manes.....	637
Robert Schneider.....	638
Leo Shuster	638
Andre Tost	639
Chris Venable.....	639

About the Contributors	641
Benjamin Carlyle	641
Robert Moores	641
Filippos Santas	642
About the Foreword Contributors	643
Massimo Pezzini	643
Roberto Medrano	643
Index.....	645

Foreword

by Massimo Pezzini

“What are the three key ingredients for successful SOA?” I was asked (in Sweden, if I remember well) by a pretty senior application architect several years ago. It was the time, circa 2004, when SOA was at the peak of what we at Gartner call “the hype cycle.” Every vendor was busily trying to reposition as a SOA player, and users were struggling to understand what SOA was and why they should care about it.

When that application architect asked me the fatal question, I had luckily already investigated SOA, especially its key “dos” and “don’ts,” for quite a while, starting in the late 1990s. I had by then spoken with quite a number of large organizations, in both North America and Europe, that had gone through the painful process of figuring out, through trial and error, how to manage a large-scale and business-critical set of SOA-based projects. Therefore, my answer was spontaneous and also came out with a rather unquestionable tone: “Discipline, discipline, and discipline!”

From my conversation with these leading-edge organizations, it was in fact pretty evident to me that what was later to be called SOA governance was a critical success factor for SOA initiatives. If you think about it for a second, this is obvious: The basic goals of SOA are

1. Reducing application development and maintenance costs, through run-time sharing of services across multiple applications
2. Increasing business agility, by effectively managing service and application life-cycle (discovery, definition, design, implementation, testing, deployment, management, maintenance, and retirement)

There is no way to achieve these goals without applying a proper set of rules and processes, which we now call SOA governance. SOA governance is in charge of making sure that services are designed and implemented to be truly reusable, that there are facilities in place (e.g., a *service repository* or *service inventory*, as it is called in this book) to enable a “reuse first” approach to application development, that ownership of (and accountability for) services is well defined and unambiguous, and that it is clear “who pays for what.” (You would be surprised to know how many SOA initiatives I analyzed came to a stalemate because of cost allocation issues....) SOA pioneers also discovered it was not sufficient to define SOA governance rules and processes. Without an organizational entity (the *SOA Center of Excellence* or *SOA Governance Program Office*, as it is called in this book) in charge of not only defining but also enabling and enforcing these rules and processes, they simply don’t happen.

You will find in this book a comprehensive and richly detailed interpretation of what these rules and processes are all about and how they can be concretely implemented. You may adopt and adapt these suggestions to your actual business and technical requirements, level of SOA maturity, organizational settings, and your company’s business and IT culture. The variety of case studies discussed in the book will also give you a sense of how concretely SOA governance can be implemented to achieve real-life business goals.

Let me conclude with a final “lesson learned” from the SOA governance trenches: Your SOA initiative may be killed by lack of governance, but too much governance can be deadly, too. Figuring out what is the “just enough” amount of governance appropriate for your company is a difficult, but worthwhile task. This book will help you accomplish that goal.

—Massimo Pezzini

VP and Research Fellow, Gartner, Inc.

Foreword

by Roberto Medrano

We have spent the better part of the last decade working on SOA governance programs at some of the world's largest and most complex IT organizations. We are very pleased, therefore, to see this important topic addressed in detail by Thomas Erl, one of this generation's truly great software architecture authors. Thomas' book is beyond timely, in our view, as it captures a serious truth that has crept up on even some of the most savvy CIOs. That is, SOA has gone from "nice to have," to "have to have," to today's reality that SOA is *just here*. Period. You have it. You don't have any choice but to have it. And now that you have, you have to govern it.

How did this happen? How did SOA emerge from the egghead shadows to become the de facto enterprise architecture across the globe? Many factors contributed to this situation, but perhaps most important has been the ascendancy of cloud computing. Though still in its infancy, cloud computing has been absolutely transformative in the role that SOA plays in day-to-day enterprise computing. The cloud is inherently service-oriented. Whether an organization is totally cloud-based, a hybrid of on-premise and cloud, or using a private cloud, its applications are now reaching out to consume and expose Web services in ways that would have been hard to imagine even a few years ago. Even organizations that shunned SOA now have one. It's called the cloud, and it's here to stay.

SOA governance and the cloud are vital companions, for better or worse. In a nutshell SOA governance is about making sure the enterprise builds the right things, build them right, and makes sure that what it has built is behaving right. With proper SOA governance, the cloud can be a strategic bonanza, smoothing the way for improving agility,

reducing risks, reducing costs and economies that everyone should want. Companies realizing the most success are those that have built a Unified SOA Governance infrastructure that governs a wide range of assets and artifacts through their entire lifecycle. Without SOA governance, the cloud threatens operational disaster and exposure to multiple levels of risk. And now, we have a thorough and well thought out book on the subject. Thomas has done the industry a great service by delving deeply into this topic in a way that readers of many different backgrounds can understand.

This book works because it gives the reader a sense of SOA governance across the full IT lifecycle and spans the organizations that are charged with managing the SOA. Thomas offers valuable insights and pragmatic tips on how to implement governance that is sensible yet effective, touching on managerial and business issues as much as technology. He probes into the nature of rules and organizations, even human nature, as he lays out the groundwork for good governance. Thomas understands that all of these aspects of governance are relevant to the success of a program. Enjoy this book. If you are involved in IT management, you will find it an indispensable companion in your quest for success with SOA.

—*Roberto Medrano*

EVP, SOA Software

Acknowledgments

Special thanks to the following reviewers who generously volunteered their time and expertise (in alphabetical order):

Mohamad Afshar
Kristofer Ågren
Randy Atkins
Jean-Paul De Baets
Toufic Boubez
Benjamin Robert Carlyle
Pethuru Chelliah
Kevin P. Davis, Ph.D.
Mike Fields
Damian Kleer
Hanu Kommalapati
Nick Laqua
Charles N. Mead, MD, MSc
David E. Michalowicz
Thomas M. Michelbach
Kam Chiu Mok
Robert Moores
Eric Roch
David S. Rogers
Filippos Santas
Mark Sigsworth
Sanjay Kumar Singh
Herbjörn Wilhelmsen
Pamela Janice Yau
Dr. Jure Zakotnik
Dr. Matthias Ziegler

This page intentionally left blank

Chapter 1



Introduction

1.1 About this Book

1.2 Recommended Reading

1.3 How this Book is Organized

1.4 Symbols, Figures, and Style Conventions

1.5 Additional Information

Imagine driving along a winding road. On the one side you have sheets of blasted rock that lead up into a mountain range, on the other side you have a steep cliff, with a freefall of several hundred feet, leading into a deep ocean. The faster you drive, the sooner you will reach your destination, but the more risky the drive. For example, you may need to swerve to avoid obstacles or adjust quickly to volatile weather conditions—risk factors that are elevated when moving at higher speeds. But, it's still tempting, because the sooner you reach that destination, the more successful your drive will be considered, by everyone.

When we design a roadmap for our SOA initiative, we lay out a direction that determines our route and a schedule that determines our rate of speed. We try to anticipate and plan for obstacles, but we know to expect the unexpected. With the necessary stakeholder support and financing in place (let's call it our "fuel in the tank"), we determine it's time to hit the road.

But before we do, let's go back to that decision point about choosing our route. A winding road with an open cliff constantly at our side represents the continuous risk of plunging over the edge, especially when maneuvering to avoid unanticipated obstacles. Such a road requires minimal work to put together and therefore a perceived opportunity to reach our goals in less time and with less expense. But, there's that risk factor we need to consider, especially of concern after we take a preliminary look over the edge to see the accumulated wreckage of the many vehicles that previously, unsuccessfully attempted this drive. We therefore reconsider.

The best analogy of IT governance I encountered was by Leo Shuster who, in his podcast interview for the International SOA + Cloud Symposium, stated that governance is like guardrails along a road. A governed roadmap is one that has, from beginning to end, controls that establish rules that we must comply with and parameters that we must function within, as we progress throughout SOA project stages.

In other words, we need to build a road with solid guardrails that keep our initiative from veering off its path. For many organizations, this realization was the result of losing significant investments to the heaps of wreckage already floating in the ocean below the cliffs of unregulated project plans. It has been a painful lesson that has, for some,

shaken their very confidence in SOA. Fortunately, out of the numerous projects and efforts that have gone into establishing SOA governance as its own field of expertise, we now have a set of proven rules and parameters that provide a stable and healthy starting point for organizations to create successful SOA governance systems.

This book is the accumulated result of many years of practice and insight provided by SOA experts, IT governance experts, and technology innovation experts. It's about the nuts and bolts of guardrail construction, maintenance, and enforcement. It's also about helping us understand that establishing a sound system of governance requires an investment and an expected return on that investment. What we put into creating those guardrails will protect the greater investment we put into the overall SOA projects that will venture down that road.

Finally, this book is about highlighting the fact that once those guardrails are in place, that governed road we built can be used over and over again, each time allowing us to drive faster, without compromising our safety. Establishing a mature system of SOA governance within our IT enterprise gives us a form of regulated agility—a robust state whereby we can rapidly respond to on-going business change without assuming unnecessary risk.

—*Thomas Erl*

1.1 About this Book

This book has a very simple objective. Its focus is solely on IT governance as it applies to the adoption of SOA and service-orientation. To that effect, it makes a clear distinction between governance and management and methodology, and then proceeds to establish a generic governance system, comprised of a series of common precepts, processes, and associated organizational roles. It further addresses governance topics that pertain to specific forms of service technology innovation, including cloud computing.

The purpose of this book is to give SOA practitioners a concrete framework that can be further augmented and extended into custom SOA governance systems and programs.

Who this Book is For

There is much discussion about the role of the SOA Governance Specialist in the upcoming chapters. While this type of IT professional will need to become an expert at everything covered in this book, the actual intended audience is much broader.

Specifically, this book will be useful to:

- IT managers and project managers that need to understand how a governance system can and should be incorporated into an SOA initiative, its impacts, requirements, and benefits.
- Architects and analysts who will be in the midst of SOA governance activities, including contribution to governance precepts and standards, as well as participation in review and audit processes.
- Enterprise architects and those involved with the authoring and maintenance of custom design standards. These individuals will be part of governance activity in almost every SOA project stage.
- Business analysts that are part of analysis teams for service modeling and for the definition of enterprise business models, such as business dictionaries, ontologies, and business processes.
- Developers, administrators, quality assurance professionals, and security specialists, who all will find themselves participating in or being affected by various SOA governance controls.
- Cloud computing professionals interested in learning about IT governance considerations specific to SOA and service-oriented solutions that encompass one or more cloud-based services or resources.

What this Book Does Not Cover

This is Not a Book About SOA Management

SOA governance has historically often been mistaken or confused with SOA management. This is a book about SOA governance only, although related management requirements and project stages are occasionally referenced. See Chapter 6 for an explanation that helps clarify the difference between governance, management, and methodology.

This is Not a Book About Cloud Computing Governance

Wherever appropriate, this book references SOA governance considerations that can pertain to cloud computing. However, it is important to note that this is not a general book about cloud computing governance—only considerations specific to applying service-orientation within cloud-based environments are mentioned. General cloud computing governance is a much broader topic that delves beyond the service level, into the various mechanisms and IT resources that can comprise cloud environments.

1.2 Recommended Reading

To further ensure that you have a clear understanding of key terms used and referenced in the upcoming chapters, you can visit the online master glossary for this book series at www.soaglossary.com to look up definitions for terms that may not be fully described in this book.

Even if you are an experienced SOA practitioner, we suggest you take the time to have a look at this online resource. A great deal of ambiguity has surrounded SOA and service-oriented computing and these explanations and definitions will ensure that you fully understand key terms and concepts in relation to this book and the book series as a whole.

Here are some recommendations for additional books that elaborate on some of the topics covered by this title:

- *SOA Principles of Service Design* – A comprehensive documentation of the service-orientation design paradigm with full descriptions of all of the principles referenced in this book.
- *SOA Design Patterns* – This is the official SOA design patterns catalog containing descriptions and examples for most of the patterns referenced in this book. You can also look up concise descriptions for these patterns at www.soapatterns.org and in Appendix D.
- *Service-Oriented Architecture: Concepts, Technology, and Design* – The coverage of service-oriented analysis and design processes in this title supplements this book with more detailed methodology-related topics.
- The title *Web Service Contract Design & Versioning for SOA* provides a great deal of technical content that may not be relevant to governance topics, except for those that aim to establish technical design and development standards. However, this book does include four chapters dedicated to Web service contract versioning topics that will be useful when dealing with governance precepts associated with the Service Versioning and Retirement project stage (see Chapter 11 and Appendix F).
- *SOA with REST* – This book documents the convergence of REST and SOA by establishing how REST services can be realized in support of service-orientation. Salient topics are reinforced with comprehensive case studies using modern REST frameworks in combination with contemporary SOA models, patterns, practices, and concepts.

For the latest information regarding these and other titles in the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*, visit www.soabooks.com.

1.3 How this Book is Organized

This book begins with Chapters 1 and 2 providing introductory content and case study background information respectively. All subsequent chapters are grouped into the following parts:

- Part I: Fundamentals
- Part II: Project Governance
- Part III: Strategic Governance
- Part IV: Appendices

Part I: Fundamentals

The first four chapters cover various introductory topics in preparation for the chapters in Parts II and III.

Chapter 3: Service-Oriented Computing Fundamentals

This chapter provides an overview of key terms and concepts associated with SOA, service-orientation, and cloud computing.

Chapter 4: SOA Planning Fundamentals

Foundational critical success factors (pillars), funding models, and basic maturity levels are described in this chapter. The “Pillars of Service-Orientation” are referenced in several subsequent chapters, especially in relation to maturity assessment and the SOA Adoption Planning project stage.

Chapter 5: SOA Project Fundamentals

This chapter provides introductory coverage of SOA project lifecycle stages, organizational roles, and the usage of service profiles. The project stages and organizational roles in particular are revisited through chapters in Parts II and III, as they relate to various SOA governance precepts and processes.

Chapter 6: Understanding SOA Governance

This must-read chapter establishes fundamental terminology and concepts pertaining to IT governance and SOA governance. Topics include an explanation of precepts and processes, the involvement of people and organizational roles, the SOA governance system, the SOA governance program, and the SOA Governance Program Office (SGPO).

Part II: Project Governance

This part of the book provides a series of chapters that step you through the SOA project lifecycle by exploring how and where various governance controls can be incorporated within different project stages. In many cases, governance controls provide entrance and exit criteria for the regulated transition from stage to stage.

Chapter 7: Governing SOA Projects

Part II begins with topics that explain how SOA project governance is approached, along with a series of overarching SOA governance precepts that apply to various project stages. This chapter concludes with a section dedicated to SOA Adoption Planning and establishes governance controls specific to this stage.

Precepts and processes covered in this chapter:

- Service Profile Standards
- SOA Governance Technology Standards
- Preferred Adoption Scope Definition
- Organizational Maturity Criteria Definition
- Standardized Funding Model
- Organizational Governance Maturity Assessment
- Adoption Impact Analysis
- Adoption Risk Assessment

Chapter 8: Governing Service Analysis Stages

A set of SOA governance controls, rules, and regulations are provided for the analysis and modeling of individual service candidates, as well as collections (or inventories) of services that need to be modeled in relation to each other.

Precepts and processes covered in this chapter:

- Service Inventory Scope Definition
- Service and Capability Candidate Naming Standards
- Service Normalization
- Service Candidate Versioning Standards
- Business Requirements Prioritization
- Service Candidate Review

Chapter 9: Governing Service Design and Development Stages

The physical design of service contracts and service architecture and logic are addressed in this chapter in relation to SOA governance precepts, processes, and organizational roles that are involved primarily to establish various standards, conventions, and compliance review processes.

Precepts and processes covered in this chapter:

- Schema Design Standards
- Service Contract Design Standards
- Service-Orientation Contract Design Standards
- SLA Template
- Service Logic Design Standards
- Service-Orientation Architecture Design Standards
- Service Logic Programming Standards
- Custom Development Technology Standards
- Service Contract Design Review
- Service Contract Registration
- Service Access Control
- Service Logic Design Review
- Legal Data Audit
- Service Logic Code Review

Chapter 10: Governing Service Testing and Deployment Stages

Quality assurance and testing activities are covered, along with steps required to deploy and maintain service implementations. For each of these topics, further governance controls and approaches are documented.

Precepts and processes covered in this chapter:

- Testing Tool Standards
- Testing Parameter Standards
- Service Testing Standards
- Cloud Integration Testing Standards
- Test Data Usage Guidelines
- Production Deployment and Maintenance Standards
- Service Test Results Review
- Service Certification Review
- Service Maintenance Review

Chapter 11: Governing Service Usage, Discovery, and Versioning Stages

We conclude this part with a look at governance topics and controls that regulate the runtime usage of services, as well as their post-implementation discovery and versioning. The Service Usage and Monitoring stage in particular is where a range of metrics are documented and further links to upcoming SOA governance vitality triggers and activities are established.

Precepts and processes covered in this chapter:

- Runtime Service Usage Thresholds
- Service Vitality Triggers
- Centralized Service Registry
- Service Versioning Strategy
- SLA Versioning Rules
- Service Retirement Notification
- Service Vitality Review

- Service Registry Access Control
- Service Registry Record Review
- Service Discovery
- Shared Service Usage Request
- Shared Service Modification Request
- Service Versioning
- Service Retirement

Part III: Strategic Governance

The next part of this book provides further governance topics that have broad or long-term applicability and are primarily relevant from a strategic perspective.

Chapter 12: Service Information and Service Policy Governance

Several additional SOA governance precepts and processes are documented in this chapter, primarily focused on the modeling, design, and standardization of business data and related models. Many of the artifacts advocated by these governance controls relate back to early SOA project stages.

Precepts and processes covered in this chapter:

- Enterprise Business Dictionary/Domain Business Dictionary
- Service Metadata Standards
- Enterprise Ontology/Domain Ontology
- Business Policy Standards
- Operational Policy Standards
- Policy Centralization
- Data Quality Review
- Communications Quality Review
- Information Alignment Audit
- Policy Conflict Audit

Chapter 13: SOA Governance Vitality

The concept of governance vitality is described in this chapter, along with explanations of common vitality triggers and vitality process activities. These are associated primarily with product service usage, but are also of strategic relevance for the on-going evolution of services and collections of services.

Chapter 14: SOA Governance Technology

This chapter begins by establishing primary categories of SOA governance technologies, and then proceeds to document common types of tools and products used to help automate various governance tasks throughout SOA project stages.

Part IV: Appendices*Appendix A: Case Study Conclusion*

This appendix provides a brief conclusion of the case study storyline.

Appendix B: Master Reference Diagrams for Organizational Roles

Throughout the chapters in Parts II and III, SOA governance precepts and processes are mapped to each other and to organizational roles within a given project stage and beyond. However, the mapping of an organization's roles is limited to a given project stage because many of the same roles are associated with multiple project stages.

This appendix provides a global, cross-project stage mapping diagram for each organizational role.

Appendix C: Service-Orientation Principles Reference

This appendix provides the profile tables (originally from *SOA Principles of Service Design*) for the service-orientation design principles referenced in this book.

Appendix D: SOA Design Patterns Reference

This appendix provides the profile tables (originally from *SOA Design Patterns*) from the official SOA design patterns catalog.

Appendix E: The Annotated SOA Manifesto

This appendix provides the annotated version of the SOA Manifesto declaration, which is also published at www.soa-manifesto.com.

Appendix F: Versioning Fundamentals for Web Services and REST Services

As a supplement for Service Versioning topics and related governance precepts, a revised version of the Fundamental Service Versioning chapter from the *Web Service Contract Design & Versioning for SOA* book is provided here, updated with new content and examples for both Web services and REST services.

Appendix G: Mapping Service-Orientation to RUP

A newly published paper that provides concrete mapping of various aspects of service-orientation with the rational unified process (RUP).

Appendix H: Additional Resources

A list of relevant Web sites and supplementary resources.

1.4 Symbols, Figures, and Style Conventions

Symbol Legend

This book contains a series of diagrams that are referred to as *figures*. The primary symbols used throughout all figures are individually described in the symbol legend located on the inside of the front cover.

Mapping Diagrams

Chapters 7 through 12 provide concrete mapping of governance controls, as follows:

- SOA governance precepts are mapped to related processes and roles
- SOA governance processes are mapped to related precepts and roles
- relevant organizational roles are mapped to related SOA governance precepts and processes specific to the current chapter

This mapping is visually illustrated via a series of diagrams, such as the one shown in Figure 1.1.

As previously explained, Appendix B further provides a series of cross-project stage mapping diagrams for organization roles.

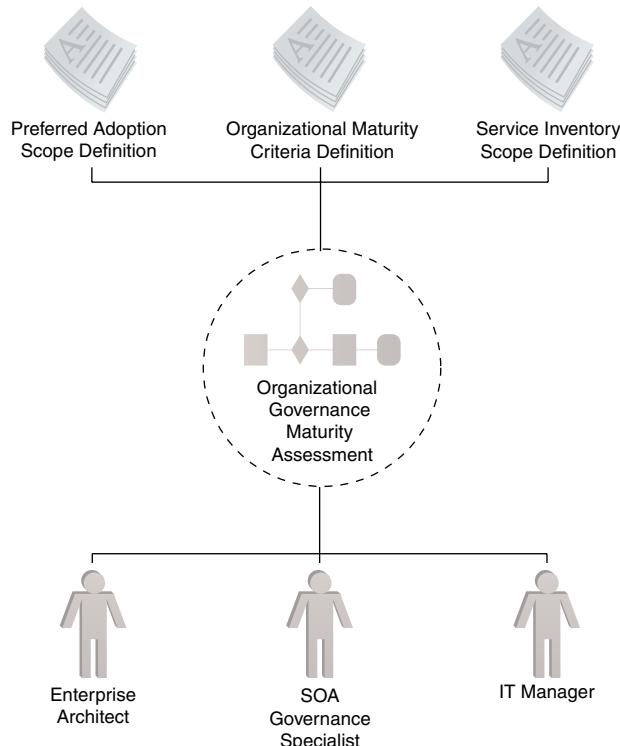


Figure 1.1

An example of a mapping diagram. The item in the center is an SOA governance process that is being mapped to three precepts (top) and three organizational roles (bottom).

SOA Principles & Patterns Sections

As a further supplement, this book occasionally references service-orientation principles and SOA patterns relevant to various governance topics. These references are generally isolated to separate *SOA Principles & Patterns* sections and provided primarily for those readers familiar with the principles and patterns covered in *SOA Principles of Service Design* and *SOA Design Patterns* series titles. Prior knowledge of these principles and patterns is not required and the corresponding references can be disregarded if they are not of interest.

Principle and pattern names are always capitalized and followed by a page number that points to their profile. Profile tables for principles are provided in Appendix C and

those for patterns are in Appendix D. In order to maintain a distinction between principles and patterns, the page number for each principle is placed in rounded parenthesis, and for patterns, square brackets are used.

For example, the following statement first references a service-orientation design principle and then an SOA design pattern:

“...the Service Loose Coupling (477) principle is supported via the application of the Decoupled Contract [517] pattern...”

Capitalization

The following are categories of topics for which terms are consistently capitalized throughout this book:

- service-orientation pillars
- SOA project stage names
- organizational roles
- organizational maturity levels
- funding models
- SOA governance precept names
- SOA governance process names

This usage of capitalization is intended to help with the identification of key terms, especially those for which mapping is provided.

1.5 Additional Information

These sections provide supplementary information and resources for the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*.

Updates, Errata, and Resources (www.soabooks.com)

Information about other series titles and various supporting resources can be found at www.soabooks.com. You are encouraged to visit this site regularly to check for content changes and corrections.

Master Glossary (www.soaglossary.com)

To avoid content overlap and to ensure constant content currency, the books in this series do not contain glossaries. Instead, a dedicated Web site at www.soaglossary.com provides a master glossary for all series titles. This site continues to grow and expand with new glossary definitions as new series titles are developed and released.

Referenced Specifications (www.soaspecs.com)

The www.soaspecs.com Web site provides a central portal to the original specification documents created and maintained by the primary standards organizations.

SOASchool.com® SOA Certified Professional (SOACP)

This text book is an official part of the SOA Certified Professional curriculum and is used in conjunction with courses and exams for the SOA Governance Specialist Certification program. The course materials that are part of this program provide additional content and lab exercises that further explore topics covered in this book.

For more information, visit www.soaschool.com.

CloudSchool.com™ Cloud Certified Professional (CCP)

Various SOA governance topics covered in this book address cloud computing considerations. Content pertaining to the governance of cloud-based services, as well as introductory content of general cloud computing topics and concepts, was provided by course material donated by the CloudSchool.com™ Cloud Certified Professional curriculum.

For more information, visit www.cloudschool.com.

The SOA Magazine (www.soamag.com)

The SOA Magazine is a regular publication provided by SOA Systems Inc. and Prentice Hall and is officially associated with the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*. *The SOA Magazine* is dedicated to publishing specialized SOA articles, case studies, and papers by industry experts and professionals. The common criteria for contributions is that each explore a distinct aspect of service-oriented computing.

Notification Service

If you'd like to be automatically notified of new book releases in this series, new supplementary content for this title, or key changes to the previously listed Web sites, use the notification form at www.soabooks.com.

Chapter 2



Case Study Background

2.1 How Case Studies are Used

2.2 Raysmoore Corporation

2.3 Case Study Continuation

2.1 How Case Studies are Used

Part II of this book provides a series of case study examples that supplement the coverage of governance topics as they pertain to different lifecycle stages of SOA projects. These examples are based on the case study background content provided by this chapter. You can readily identify case study content by looking for sections with a light gray shading.

2.2 Raysmoore Corporation

Following the unexpected rise of precious metal values, the international mining industry has witnessed a dramatic increase in demand, production, and competition. Raysmoore is a five-year old mining corporation that has steadily established itself as a top producer of fine and coarse gold and silver. It has an internal supply chain that reaches directly into the retail sector where precious stones are sold as part of jewelry, watches, and other retail goods.

History

Two years ago, Raysmoore was comprised of a head office and three mining sites within North America. As a result of the strategic acquisition of Lovelt Inc. and Reeldrill Ltd. (two off-shore mining companies), Raysmoore has grown by almost 50%. Due to the specialized nature of the acquired companies, both Lovelt and Reeldrill remain as relatively independent subsidiaries within the overall Raysmoore corporate umbrella.

As a result of its increased size and multi-cultural market presence, Raysmoore has had to comply with a number of new regulations. Lovelt mining locations are based in South America and Reeldrill mining sites are scattered throughout Central America. Raysmoore management has noticed that governmental policies change more frequently and unexpectedly in these regions.

IT Environment

Raysmoore has a sizeable and diversified IT environment with a number of centralized systems, including a financials system, an enterprise resource planning (ERP) platform,

and a separate human resources application. It further relies on its subsidiaries' proprietary systems to perform specialized business functions that require access to cross-domain business data. As a result, various forms of functional overlap exist throughout the Raysmore ecosystem.

Raysmore's technology stack includes a diverse set of legacy hardware and software. It has a sizeable AS/400 installation that houses legacy resources, some of which were purchased while others were custom-developed. As qualified AS/400 support began to erode, legacy applications were either migrated to distributed platforms, or maintained "as-is" with little or no changes. Raysmore further has a large base of UNIX and Windows technologies that include its financials and HR systems.

Raysmore has always been conservative in its technology evaluation and selection. Raysmore's central IT group oversees its IT enterprise, as well as those from Lovelt and Reeldrill.

Business Goals and Obstacles

Raysmore would like to keep growing. The CEO's growth strategy continues to be acquisition oriented, and the company's central functions (including IT) have therefore been mandated to become agile and flexible in order to minimize the impact of the integration of new assets.

In response to the agility mandate, the CIO that oversees the Raysmore, Lovelt, and Reeldrill IT divisions recently kicked off a large-scale SOA initiative aimed at service-enabling their global IT enterprise. A primary objective of this effort is to integrate the respective divisions so as to span the supply chain across all business domains.

The project has been slow to get off the ground largely due to the company's size and political landscape. Raysmore's subsidiaries have their own IT departments that operate in isolation, and have demanded to retain authority over their respective domains.

To make matters more challenging, Raysmore IT has been asked to cut costs by 10% across the board. As a result, the CIO is considering reducing the planned amount of on-premise IT infrastructure technologies required for the SOA adoption project and instead investigate leasing options with third-party cloud providers.

With the pending complexities of a potentially expensive and pivotal SOA adoption project, the CIO decides that nothing will proceed until a solid governance system is in place.

2.3 Case Study Continuation

Additional case study content is provided throughout the following chapters:

- Chapter 7: *Governing SOA Projects*
- Chapter 8: *Governing Service Analysis Stages*
- Chapter 9: *Governing Service Design and Development Stages*
- Chapter 10: *Governing Service Testing and Deployment Stages*
- Chapter 11: *Governing Service Usage, Discovery, and Versioning Stages*

Sections in these chapters within Part II of this book are supplemented with sample case study scenarios and examples pertaining to the covered topic areas.



Part I

Fundamentals

Chapter 3: Service-Oriented Computing Fundamentals

Chapter 4: SOA Planning Fundamentals

Chapter 5: SOA Project Fundamentals

Chapter 6: Understanding SOA Governance

This page intentionally left blank

A black and white photograph showing a close-up view of a window or door covered by horizontal blinds. The blinds are partially open, creating a series of parallel diagonal lines that lead the eye towards the right side of the frame. The lighting is dramatic, with strong highlights and shadows on the blinds and the surrounding wall.

Chapter 3

Service-Oriented Computing Fundamentals

3.1 Basic Terminology

3.2 Further Reading

This chapter describes fundamental terms and concepts associated with service-oriented computing, including those related to service-oriented architecture, service-orientation, and cloud computing.

3.1 Basic Terminology

Upcoming sections provide definitions for the following terms:

- Service-Oriented Computing
- Service-Orientation
- Service-Oriented Architecture (SOA)
- Services
- SOA Manifesto
- Cloud Computing
- IT Resources
- Cloud
- On-Premise
- Cloud Deployment Models
- Cloud Consumers and Cloud Providers
- Cloud Delivery Models
- Service Models
- Service Composition
- Service Inventory
- Service Portfolio
- Service Candidate
- Service Contract

- Service-Related Granularity
- SOA Design Patterns

These terms are used throughout this book.

Service-Oriented Computing

Service-oriented computing is an umbrella term that represents a new generation distributed computing platform. As such, it encompasses many things, including its own design paradigm and design principles, design pattern catalogs, pattern languages, a distinct architectural model, and related concepts, technologies, and frameworks.

Service-orientation (explained shortly) emerged as a formal method in support of achieving the following goals and benefits associated with service-oriented computing:

- *Increased Intrinsic Interoperability* – Services within a given boundary are designed to be naturally compatible so that they can be effectively assembled and reconfigured in response to changing business requirements.
- *Increased Federation* – Services establish a uniform contract layer that hides underlying disparity, allowing them to be individually governed and evolved.
- *Increased Vendor Diversification Options* – A service-oriented environment is based on a vendor-neutral architectural model, allowing the organization to evolve the architecture in tandem with the business without being limited only to proprietary vendor platform characteristics.
- *Increased Business and Technology Domain Alignment* – Some services are designed with a business-centric functional context, allowing them to mirror and evolve with the business of the organization.
- *Increased ROI* – Most services are delivered and viewed as IT assets that are expected to provide repeated value that surpasses the cost of delivery and ownership.
- *Increased Organizational Agility* – New and changing business requirements can be fulfilled more rapidly by establishing an environment in which solutions can be assembled or augmented with reduced effort by leveraging the reusability and native interoperability of existing services.
- *Reduced IT Burden* – The enterprise as a whole is streamlined as a result of the previously described goals and benefits, allowing IT itself to better support the organization by providing more value with less cost and less overall burden.

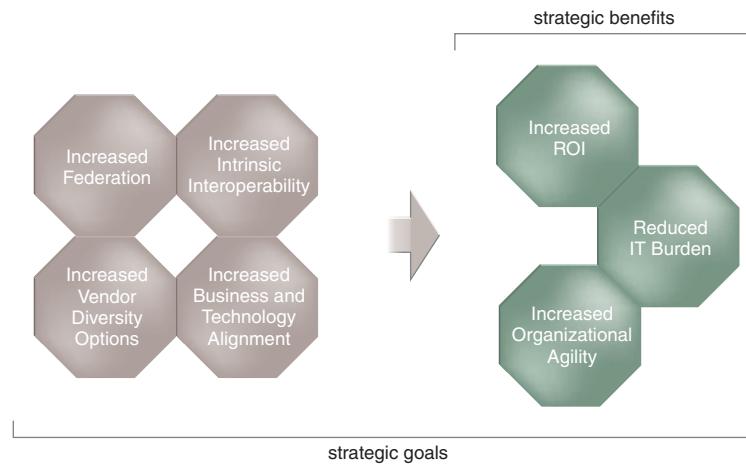


Figure 3.1

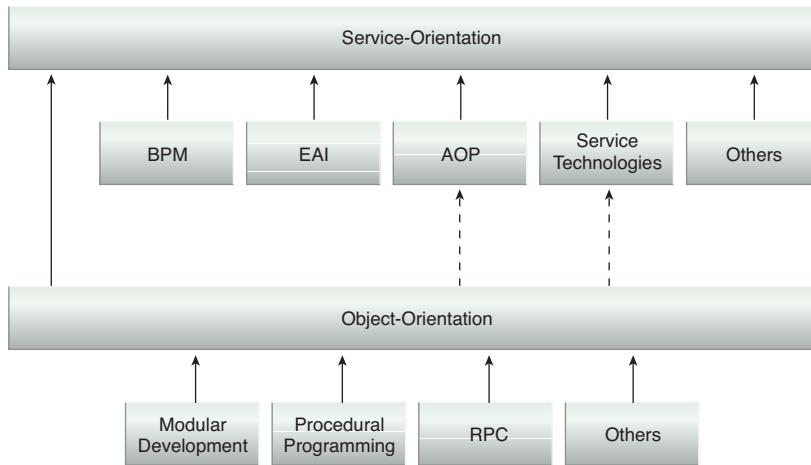
The latter three goals listed in the previous bulleted list represent target strategic benefits that are achieved when attaining the first four goals.

Note that these strategic goals are also commonly associated with SOA, as explained in the *SOA Manifesto* section.

Service-Orientation

Service-orientation is a design paradigm intended for the creation of solution logic units that are individually shaped so that they can be collectively and repeatedly utilized in support of the realization of the specific strategic goals and benefits associated with service-oriented computing.

Solution logic designed in accordance with service-orientation can be qualified with “service-oriented,” and units of service-oriented solution logic are referred to as “services.” As a design paradigm for distributed computing, service-orientation can be compared to object-orientation (or object-oriented design). Service-orientation, in fact, has many roots in object-orientation (as first documented in Chapter 14 of *SOA Principles of Service Design*) and has also been influenced by other industry developments.

**Figure 3.2**

Service-orientation is very much an evolutionary design paradigm that owes much of its existence to established design practices and technology platforms.

The service-orientation design paradigm is primarily comprised of eight specific design principles:

- *Standardized Service Contract* – “Services within the same service inventory are in compliance with the same contract design standards.”
- *Service Loose Coupling* – “Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment.”
- *Service Abstraction* – “Service contracts only contain essential information and information about services is limited to what is published in service contracts.”
- *Service Reusability* – “Services contain and express agnostic logic and can be positioned as reusable enterprise resources.”
- *Service Autonomy* – “Services exercise a high level of control over their underlying runtime execution environment.”
- *Service Statelessness* – “Services minimize resource consumption by deferring the management of state information when necessary.”
- *Service Discoverability* – “Services are supplemented with communicative metadata by which they can be effectively discovered and interpreted.”
- *Service Composability* – “Services are effective composition participants, regardless of the size and complexity of the composition.”

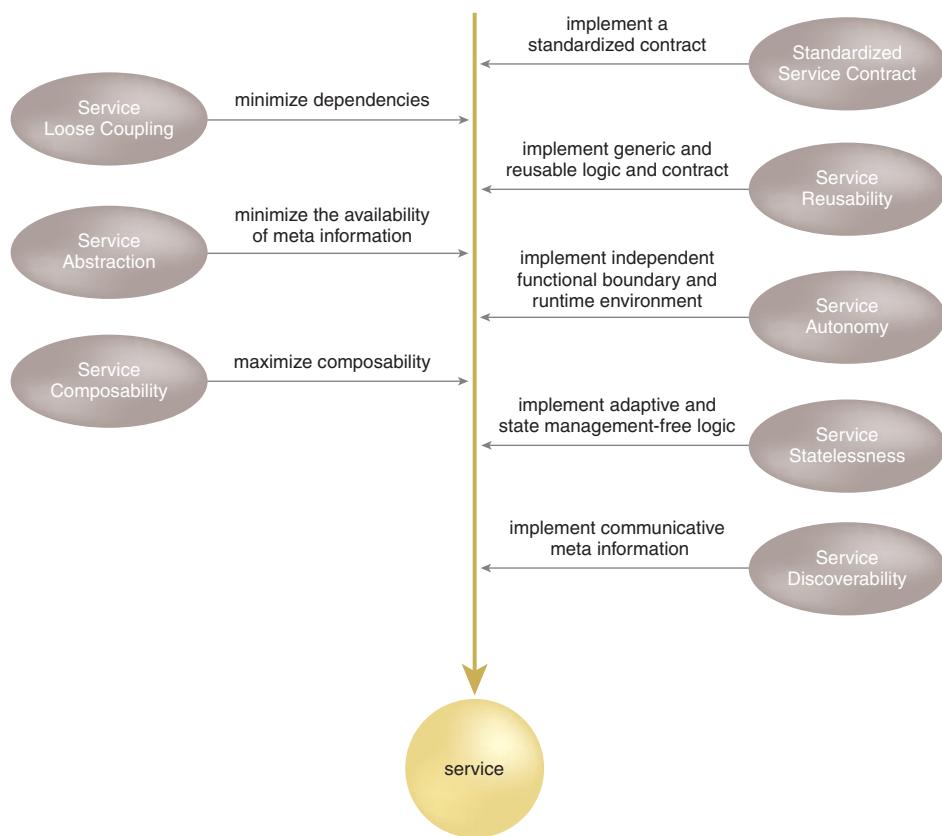


Figure 3.3

The principles on the left have a regulatory influence, whereas the application of the principles on the right primarily results in concrete characteristics being established within the service architecture.

Service-Oriented Architecture (SOA)

Service-oriented architecture is a technology architectural model for service-oriented solutions with distinct characteristics in support of realizing service-orientation and the strategic goals associated with service-oriented computing.

The four base characteristics we look to establish in any form of SOA are:

- *Business-Driven* – The technology architecture is aligned with the current business architecture. This context is then constantly maintained so that the technology architecture evolves in tandem with the business over time.
- *Vendor-Neutral* – The architectural model is not based solely on a proprietary vendor platform, allowing different vendor technologies to be combined or replaced over time in order to maximize business requirements fulfillment on an on-going basis.
- *Enterprise-Centric* – The scope of the architecture represents a meaningful segment of the enterprise, allowing for the reuse and composition of services and enabling service-oriented solutions to span traditional application silos.
- *Composition-Centric* – The architecture inherently supports the mechanics of repeated service aggregation, allowing it to accommodate constant change via the agile assembly of service compositions.

These characteristics collectively define the fundamental requirements a technology architecture must fulfill to be fully supportive of service-orientation.

As a form of technology architecture, an SOA implementation can consist of a combination of technologies, products, APIs, supporting infrastructure extensions, and various other parts. The actual complexion of a deployed service-oriented architecture is unique within each enterprise; however, it is typified by the introduction of new technologies and platforms that specifically support the creation, execution, and evolution of service-oriented solutions. As a result, building a technology architecture around the service-oriented architectural model establishes an environment suitable for solution logic that has been designed in compliance with service-orientation design principles.

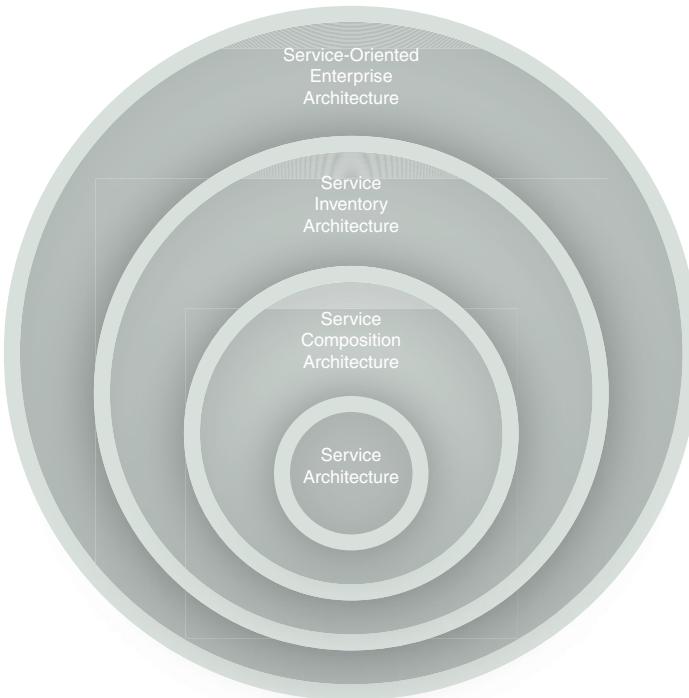


Figure 3.4

The layered SOA model establishes the four common SOA types: service architecture, service composition architecture, service inventory architecture, and service-oriented enterprise architecture. (These different architectural types, along with the four SOA characteristics, are explained in detail in the book *SOA Design Patterns*.)

NOTE

Let's briefly recap the previous three terms to clearly establish how they relate to each other and specifically how they lead to a definition of SOA:

- There is a set of strategic goals associated with service-oriented computing.
- These goals represent a specific target state.
- Service-orientation is the paradigm that provides a proven method for achieving this target state.
- When we apply service-orientation to the design of software, we build units of logic called “services.”

- Service-oriented solutions are comprised of one or more services.
- To build successful service-oriented solutions, we need a distributed technology architecture with specific characteristics.
- These characteristics distinguish the technology architecture as being service-oriented. This is SOA.

Services

A *service* is a unit of logic to which service-orientation has been applied to a meaningful extent. It is the application of service-orientation design principles that distinguishes a unit of logic as a service compared to units of logic that may exist solely as objects, components, Web services, REST services, and/or cloud-based services.

Subsequent to conceptual service modeling, design, and development stages implement a service as a physically independent software program with specific design characteristics that support the attainment of the strategic goals associated with service-oriented computing.

Each service is assigned its own distinct functional context and is comprised of a set of capabilities related to this context. Therefore, a service can be considered a container of capabilities associated with a common purpose (or functional context).

It is important to view and position SOA and service-orientation as being neutral to any one technology platform. By doing so, you have the freedom to continually pursue the strategic goals associated with service-oriented computing by leveraging on-going service technology advancements.

Any implementation technology that can be used to create a distributed system may be suitable for the application of service-orientation. Three common service implementation mediums currently exist: components, Web services, and REST services. Any of these forms of service implementations can also exist as cloud-based services (as explained in the upcoming *Cloud* section).



Figure 3.5

The chorded circle symbol is used to represent a service, primarily from a contract perspective.

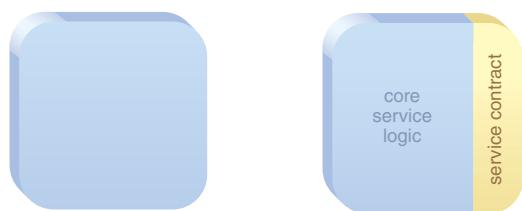
Services as Components

A *component* is a software program designed to be part of a distributed system. It provides a technical interface comparable to a traditional application programming interface (API) through which it exposes public capabilities as *methods*, thereby allowing it to be explicitly invoked by other programs.

Components have typically relied on platform-specific development and runtime technologies. For example, components can be built using Java or .NET tools and are then deployed in a runtime environment capable of supporting the corresponding component communications technology requirements, as implemented by the chosen development platform.

Figure 3.6

The symbols used to represent a component. The symbol on the left is a generic component that may or may not have been designed as a service, whereas the symbol on the right is labeled to indicate that it has been designed as a service.



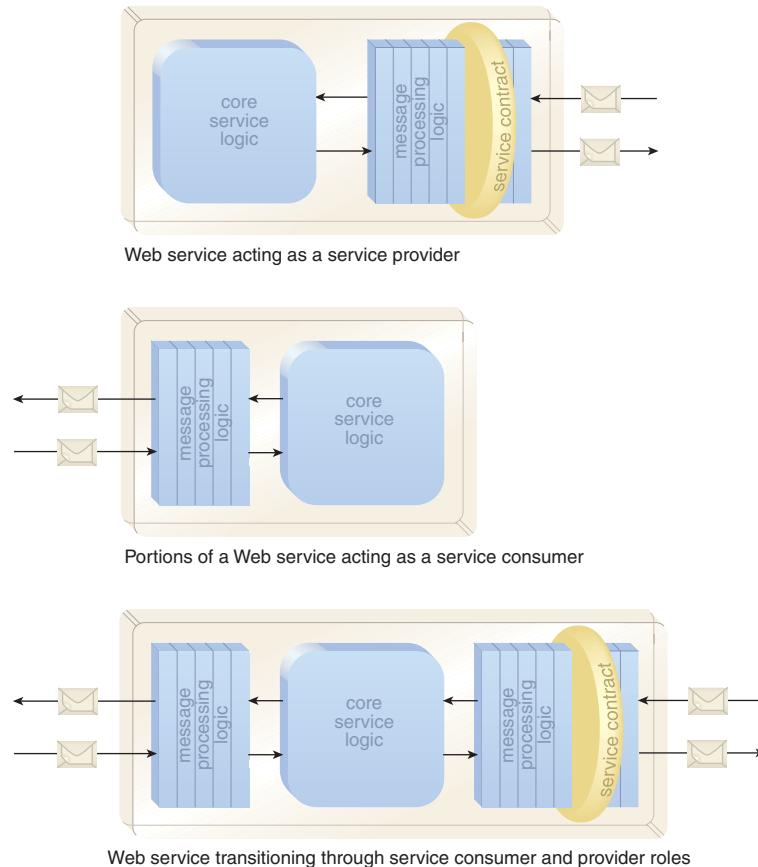
NOTE

Building service-oriented components is one of the topics covered in the books *SOA with .NET & Windows Azure* and *SOA with Java*, both titles in the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*.

Services as Web Services

A *Web service* is a body of solution logic that provides a physically decoupled technical contract consisting of a WSDL definition, one or more XML Schema definitions and also possible WS-Policy expressions. The Web service contract exposes public capabilities as *operations*, establishing a technical interface but without any ties to a proprietary communications framework.

Service-orientation can be applied to the design of Web services. Web services provide an architectural model whereby the service contract is physically decoupled and vendor-neutral. This is conducive to several of the design goals associated with service-orientation.

**Figure 3.7**

Three variations of a single Web service showing the different physical parts of its architecture that come into play, depending on the role it assumes at runtime.

NOTE

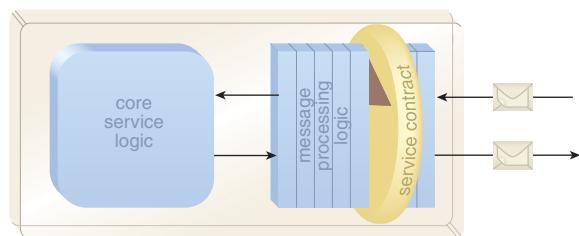
Coverage of Web services in relation to SOA is provided by the series' titles *Web Service Contract Design and Versioning for SOA* and *Service-Oriented Architecture: Concepts, Technology, and Design*.

Services as REST Services

REST services (or RESTful services) are designed in compliance with the REST architectural style. A REST service architecture focuses on the resource as the key element of abstraction, with an emphasis on simplicity, scalability, and usability. REST services can be further shaped by the application of service-orientation principles.

Figure 3.8

A REST service, depicted similar to a Web service, except for the service contract symbol that indicates the service is accessed via a uniform contract.



NOTE

How REST services can be designed in support of SOA and service-orientation is explored in the book *SOA with REST*, as part of the *Prentice Hall Service-Oriented Computing Series* from Thomas Erl. Note also that this book has been supplemented with new versioning content specific for REST services. This content can be found in Appendix F.

SOA Manifesto

Historically, the term “service-oriented architecture” (or “SOA”) has been used so broadly by the media and within vendor marketing literature that it has almost become synonymous with service-oriented computing itself. *The SOA Manifesto* (published at www.soa-manifesto.org) is a formal declaration authored by a diverse working group comprised of industry thought leaders during the 2nd International SOA Symposium in Rotterdam in 2009. This document establishes, at a high level, a clear separation of service-oriented architecture and service-orientation in order to address the ambiguity that had been causing confusion in relation to the meaning of the term “SOA.”

The Annotated SOA Manifesto is published at www.soa-manifesto.com and in Appendix E of this book. This version of the SOA Manifesto is recommended reading as it elaborates on statements made in the original SOA Manifesto.

Cloud Computing

Cloud computing is a specialized form of distributed computing that introduces utilization models for remotely provisioning scalable and measured IT resources. The primary benefits associated with cloud computing are:

- *Reduced Investment and Proportional Costs* – Cloud consumers that use cloud-based IT resources can generally lease them with a pay-for-use model, which allows them to pay a usage fee for only the amount of the IT resource actually used, resulting in directly proportional costs. This gives an organization access to IT resources without having to purchase its own, resulting in reduced investment requirements. By lowering required investments and incurring costs that are proportional to their needs, cloud consumers can scale their IT enterprise effectively and proactively.
- *Increased Scalability* – IT resources can be flexibly acquired from a cloud provider, almost instantaneously and at a wide variety of usage levels. By scaling with cloud-based IT resources, cloud consumers can leverage this flexibility to increase their responsiveness to planned and unforeseen changes.
- *Increased Availability and Reliability* – Cloud providers generally offer resilient IT resources for which they are able to guarantee high levels of availability. Cloud environments can be based on a modular architecture that provides extensive failover support to further increase reliability. Cloud consumers that lease access to cloud-based IT resources can therefore benefit from increased availability and reliability.

When appropriate, these benefits can help realize the strategic goals of service-oriented computing by extending and enhancing service-oriented architectures and increasing the potential of realizing certain service-orientation principles.

IT Resources

An *IT resource* is a broad term to refer to any physical or virtual IT-related artifact (software or hardware). For example, a physical server, a virtual server, a database, and a service implementation are all forms of IT resources.

Even though a service is considered an IT resource, it is important to acknowledge that a service architecture will commonly encapsulate and connect to other IT resources. This distinction is especially important in cloud-based environments, where a cloud service

is classified as a remotely accessible IT resource that may encompass and depend on various additional cloud-based IT resources that are only accessible from within the cloud.

NOTE

The Cloud Resource Administrator role described in Chapter 5 can be involved with cloud service administration and the administration of internal cloud-based IT resources.

Cloud

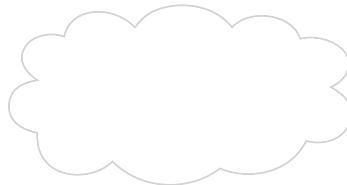
A *cloud* is a distinct IT environment designed for the purpose of remotely provisioning scalable and measured IT resources. In order to remotely provision scalable and measured IT resources in an effective manner, an IT environment requires a specific set of characteristics. These characteristics need to exist to a meaningful extent for the IT environment to be considered an effective cloud.

- *On-Demand Usage* – This characteristic allows for the usage of self-provisioned IT resources to be automated so that no further human involvement by the provider or consumer of these resources is required.
- *Ubiquitous Access* – This is the ability for a cloud-based IT resource to be widely accessible.
- *Multitenancy* – Multitenancy is the characteristic of a software program that enables an instance of the program to serve different consumers (tenants), each of which is isolated from the other.
- *Elasticity* – This is the ability of a cloud to enable its consumers to scale cloud-based IT resources up or down, as required.
- *Measured Usage* – This represents the ability of a cloud platform to keep track of the usage of its IT resources by its consumers.
- *Resilient Computing* – This characteristic represents a form of failover that distributes redundant implementations of IT resources across physical locations.

Services that reside in cloud environments are referred to as *cloud services* or *cloud-based services*. Services are one form of IT resource hosted by clouds.

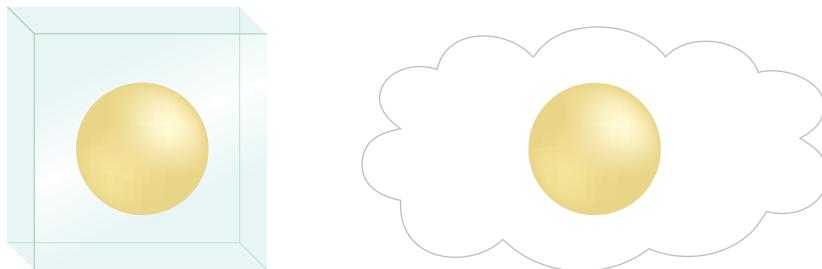
Figure 3.9

The symbol used to represent a cloud environment.



On-Premise

The term *on-premise* is used as a qualifier to indicate that a service or IT resource (or some other artifact) resides within an internal IT enterprise environment, as opposed to a cloud-based environment. The use of “on-premise” within this book is primarily associated with discussions pertaining to cloud computing issues. By default, coverage of services, architectures, infrastructure, and other types of implementations are assumed to be on-premise, unless otherwise qualified.

**Figure 3.10**

A service shown on-premise, within an IT enterprise (left) and a cloud-based service, deployed within a cloud (right).

Cloud Deployment Models

A *cloud deployment model* represents a specific type of cloud environment, primarily distinguished by ownership and size.

There are four common deployment models:

- *Public Cloud* – A public cloud is a publically accessible cloud environment owned by a third-party cloud provider.
- *Community Cloud* – A community cloud is similar to a public cloud except that its access is limited to a specific community of cloud consumers.

- *Private Cloud* – A private cloud is owned by a single organization.
- *Hybrid Cloud* – A hybrid cloud is a cloud environment of two or more different cloud deployment models.

Variations of these models can also exist.

Cloud Consumers and Cloud Providers

The organization that uses cloud services is referred to as the *cloud consumer*, whereas the organization that owns and offers cloud IT resources and services is the *cloud provider*. With private clouds the cloud consumer and cloud provider can be the same organization, in which case these roles are assumed by departments or groups within the organization.

Cloud Delivery Models

A *cloud delivery model* represents a specific combination of IT resources offered by a cloud provider.

Three common delivery models are used:

- *Infrastructure-as-a-Service (IaaS)* – The IaaS delivery model provides a self-contained IT environment comprised of infrastructure-centric IT resources.
- *Platform-as-a-Service (PaaS)* – The PaaS delivery model provides a pre-defined cloud environment with already deployed and configured IT resources suitable for the development and deployment of applications.
- *Software-as-a-Service (SaaS)* – The SaaS delivery model generally represents a product that exists as a shared cloud service offered by a cloud provider to cloud consumers.

Variations of these models can also exist.

Service Models

A *service model* is a classification used to indicate that a service belongs to one of several predefined types based on the nature of the logic it encapsulates, the reuse potential of this logic, and how the service may relate to domains within its enterprise.

The following three service models are common to most enterprise environments and therefore common to most SOA projects:

- *Task Service* – A service with a non-agnostic functional context that generally corresponds to single-purpose, parent business process logic. A task service will usually encapsulate the composition logic required to compose several other services in order to complete its task.
- *Entity Service* – A reusable service with an agnostic functional context associated with one or more related business entities (such as invoice, customer, claim, etc.). For example, a Purchase Order service has a functional context associated with the processing of purchase order-related data and logic.
- *Utility Service* – Also a reusable service with an agnostic functional context, but this type of service is intentionally not derived from business analysis specifications and models. It encapsulates low-level technology-centric functions, such as notification, logging, and security processing.

NOTE

Due to their reuse potential, both entity and utility services are considered *shared services* (also referred to as *enterprise resources*). Most of the governance topics covered in this book pertain to the design-time and runtime regulation of shared services.

Service models play an important role during service-oriented analysis and service-oriented design phases. Although the just listed service models are well established, it is not uncommon for an organization to create its own service models. Often these new classifications tend to be derived from one of the aforementioned fundamental service models.

Agnostic Logic and 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 specific to single purpose tasks is intentionally omitted, agnostic logic is considered multi-purpose. On the flipside, logic that is specific to (contains knowledge of) a single-purpose task is labeled as *non-agnostic* logic.

Another way of thinking about agnostic and non-agnostic logic is to focus on the extent to which the logic can be repurposed. Because agnostic logic is expected to be multi-purpose, it has reuse potential and therefore forms the basis of shared service logic. Once reusable, this logic is truly multi-purpose in that it, as a single software program (or service), can be used to automate multiple business processes.

Non-agnostic logic does not have these types of expectations. It is deliberately designed as a single-purpose software program (or service) and therefore has different characteristics and requirements.

Service Composition

A *service composition* is an aggregate of services collectively composed to automate a particular task or business process. To qualify as a composition, at least two participating services plus one composition initiator need to be present. Otherwise, the service interaction only represents a point-to-point exchange.

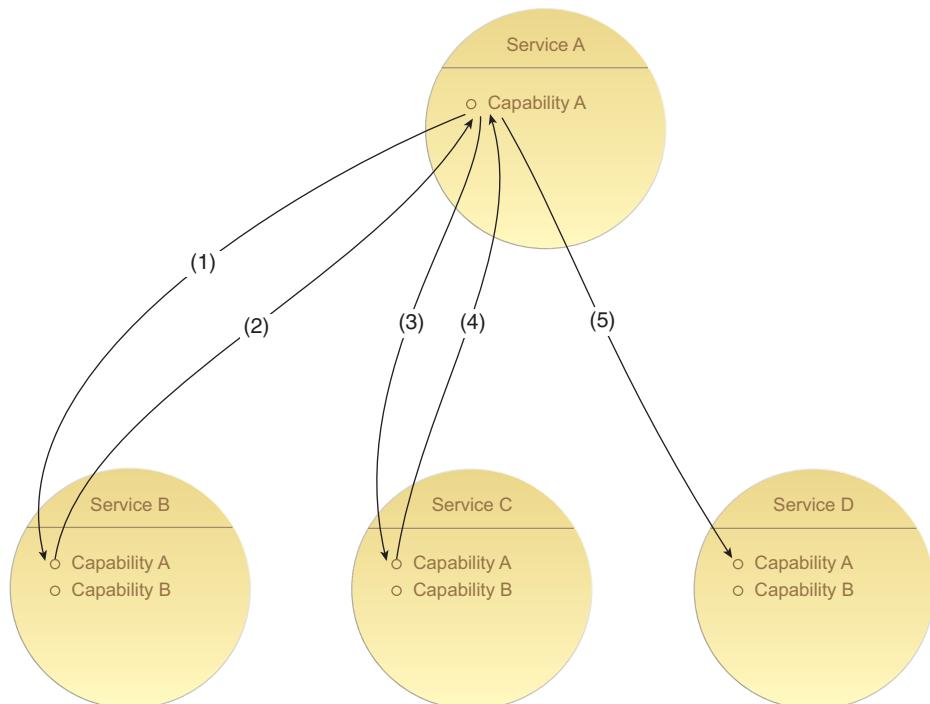


Figure 3.11

A service composition comprised of four services. The arrows indicate a sequence of modeled message exchanges. Note arrow #5 representing a one-way, asynchronous data delivery from Service A to Service D.

Service compositions can be classified into primitive and complex variations. In early service-oriented solutions, simple logic was generally implemented via point-to-point exchanges or primitive compositions. As the surrounding technology matured, complex compositions became more common.

Much of the service-orientation design paradigm revolves around preparing services for effective participation in numerous complex compositions—so much so that the Service Composability design principle exists, dedicated solely to ensuring that services are designed in support of repeatable composition.

Service Inventory

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. When an organization has multiple service inventories, this term is further qualified as *domain service inventory*.

Service inventories are typically created through top-down delivery processes that result in the definition of *service inventory blueprints*. The subsequent application of service-orientation design principles and custom design standards throughout a service inventory is of paramount importance so as to establish a high degree of native inter-service interoperability. This supports the repeated creation of effective service compositions in response to new and changing business requirements.

Service Portfolio

Service portfolio (also commonly referred to as a “service catalog”) is a separate term used to represent a set of the services within a given IT enterprise. The distinction between service inventory and service portfolio is important as these and related terms are used within different contexts, as follows:

- Service inventory represents a collection of implemented services that are independently owned and governed.
- The Service Inventory Analysis is a modeling process by which service candidates are defined for a new or existing service inventory.
- A service inventory blueprint is a technical specification that represents the result of having performed a service inventory analysis. Subsequent iterations of the service inventory analysis process can expand or further refine a service inventory blueprint.

- The term “service portfolio” has a less specific definition than service inventory in that it can represent all or a subset of the services within an IT enterprise.
- A service portfolio often exists as a high-level documentation of services used for planning purposes.
- A service portfolio most commonly encompasses one or multiple service inventories.

Service portfolio management is the practice of planning the definition, delivery, and evolution of collections of services.

Service Candidate

When conceptualizing services during the service-oriented analysis and service modeling processes, services are defined on a preliminary basis and still subject to a great deal of change and refinement before they are handed over to the Service-Oriented Design project stage responsible for producing physical service contracts. The term *service candidate* is used to help distinguish a conceptualized service from an actual implemented service.

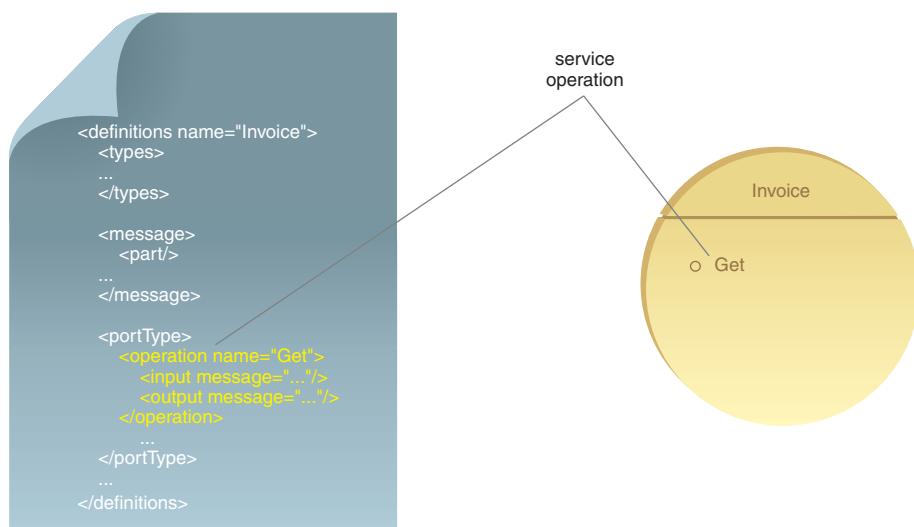


Figure 3.12

The Get operation (right) is first modeled and then forms the basis of the actual operation definition within a WSDL document (left).

Service Contract

A *service contract* is comprised of one or more published documents that express meta information about a service. The fundamental part of a service contract consists of the documents that express its technical interface. These form the technical service contract, which essentially establishes an API into the functionality offered by the service via its capabilities.

When services are implemented as Web services, the most common service description documents are the WSDL definition, XML Schema definition, and WS-Policy definition. A Web service generally has one WSDL definition, which can link to multiple XML Schema and policy definitions. When services are implemented as components, the technical service contract is comprised of a technology-specific API.

Services implemented as REST services are commonly accessed via a uniform contract, such as the one provided by HTTP. Service contracts are depicted differently depending on whether a uniform contract is involved.

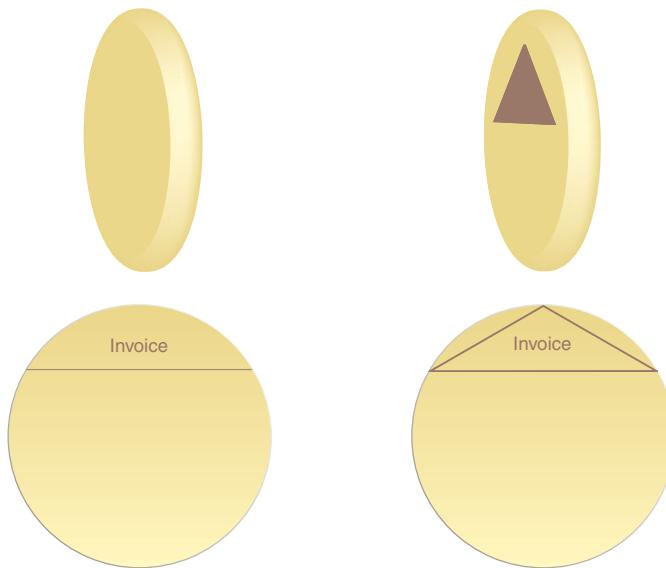


Figure 3.13

The standard symbol used to display a service contract (left) and one that is accessed via a uniform contract (right).

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

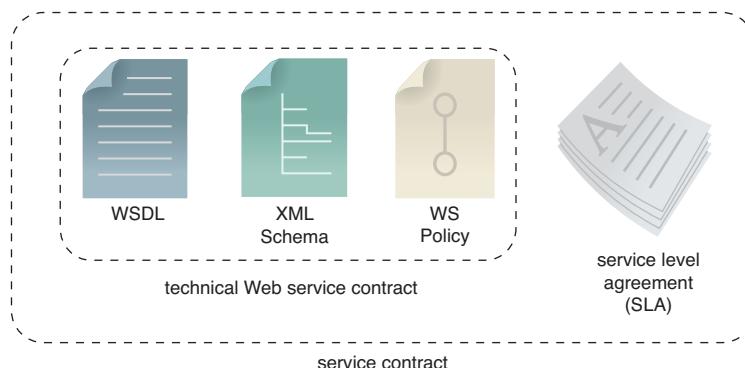


Figure 3.14

The common documents that comprise the technical Web service contract, plus a human-readable SLA.

Within service-orientation, the design of the service contract is of paramount importance—so much so, that the Standardized Service Contract (475) design principle and the aforementioned service-oriented design process are dedicated solely to the standardized creation of service contracts.

NOTE

Service contract design and versioning for Web services is a topic specifically covered in the book *Web Service Contract Design & Versioning for SOA*, as part of this series.

Service-Related Granularity

When designing services, there are different granularity levels that need to be taken into consideration, as follows:

- *Service Granularity* – This level of granularity represents the functional scope of a service. For example, fine-grained service granularity indicates that there is little logic associated with the service's overall functional context.

- *Capability Granularity* – The functional scope of individual service capabilities (operations) is represented by this granularity level. For example, a GetDetail capability will tend to have a finer measure of granularity than a GetDocument capability.
- *Constraint Granularity* – The level of validation logic detail is measured by constraint granularity. The more coarse constraint granularity is, the less constraints (or smaller the amount of validation logic) a given capability will have.
- *Data Granularity* – This granularity level represents the quantity of data processed. For example, from a Web service contract perspective, this corresponds to input, output, and fault messages. A fine level of data granularity is equivalent to a small amount of data.

Because the level of service granularity determines the functional scope of a service, it is usually determined during analysis and modeling stages that precede service contract design. Once a service's functional scope has been established, the other granularity types come into play and affect both the modeling and physical design of a service contract.

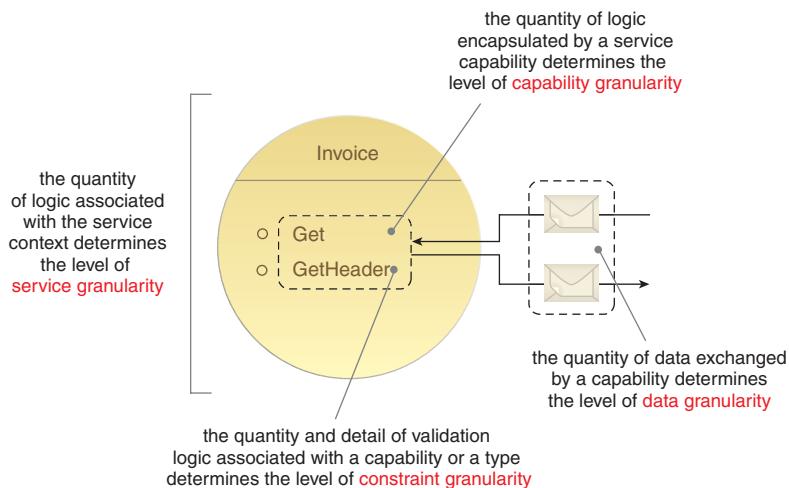


Figure 3.15

The four granularity levels that represent various characteristics of a service and its contract. Note that these granularity types are, for the most part, independent of each other.

SOA Design Patterns

A design pattern is a proven solution to a common design problem. The *SOA design pattern catalog* provides a collection of design patterns that provide practices and techniques for solving common problems in support of service-orientation.

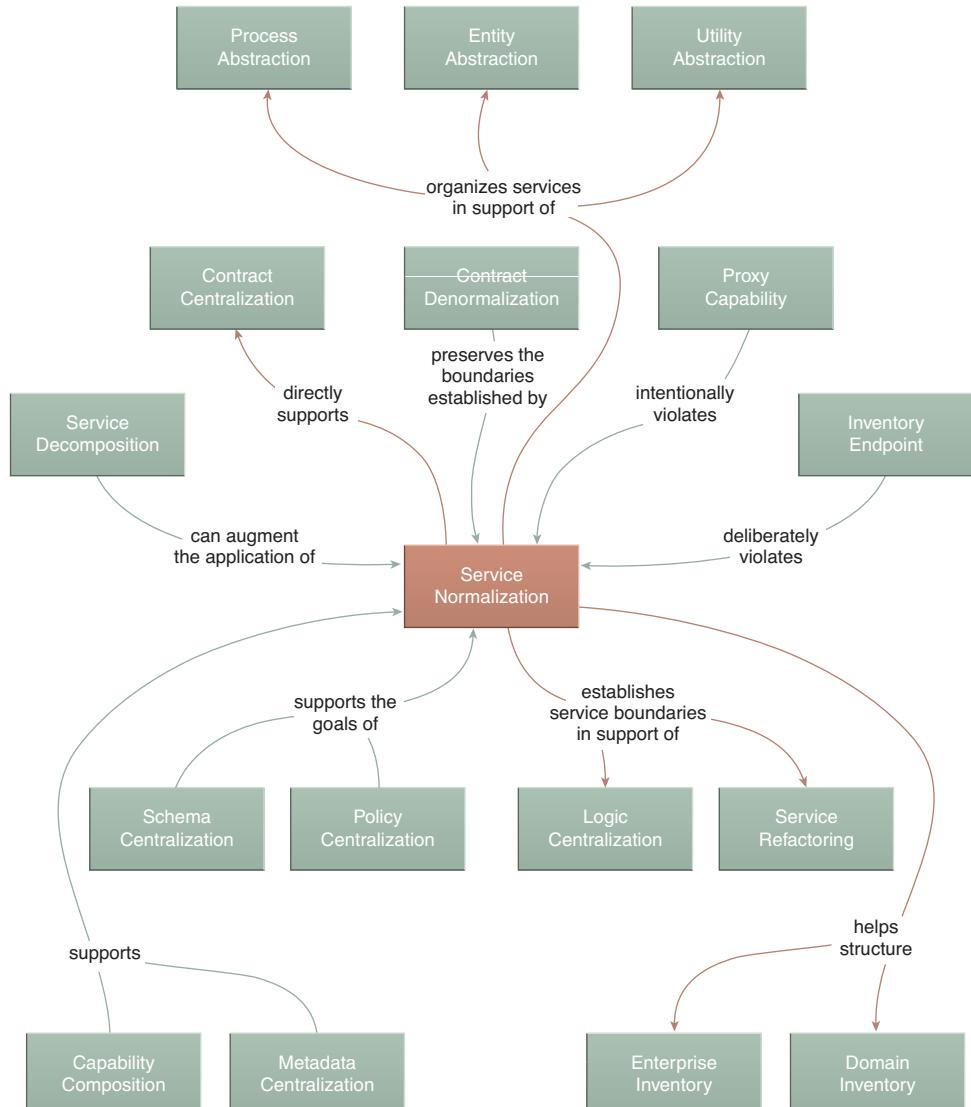


Figure 3.16

SOA design patterns form a design pattern language that allows patterns to be applied in different combinations and in different sequences in order to solve various complex design problems.

SOA design patterns do not only pertain to the actual design of services and related environments. Many patterns represent solutions that are realized by the application of standards and technologies highly relevant to SOA governance. Throughout this book you will notice references to patterns as a supplemental resource that further formalizes a given solution. In some cases, an SOA design pattern is even the primary basis of an SOA governance precept.

NOTE

Profiles of SOA design patterns are provided in Appendix D of this book and are also published online at www.soapatterns.org.

3.2 Further Reading

- Explanations of the service-oriented computing goals and benefits are available at www.whatissoa.com and in Chapter 3 of *SOA Principles of Service Design*.
- Explanations and definitions of concepts and terminology pertaining to cloud computing are available at www.whatiscloud.com.
- For information about SOA types and the distinct characteristics of the service-oriented technology architecture, see Chapter 4 of *SOA Design Patterns*.
- Design principles are referenced throughout this book but represent a separate subject matter that is covered in *SOA Principles of Service Design*. Introductory coverage of service-orientation as a whole is also available at www.soaprinciples.com and all eight principle profile tables are provided in Appendix C of this book.
- For a comparison of service-orientation and object-orientation concepts and principles, see Chapter 14 in *SOA Principles of Service Design*.
- Numerous design patterns are referenced in the upcoming chapters. These patterns are part of a greater SOA design patterns catalog that was published in the book *SOA Design Patterns*. Pattern profiles are available online at the SOAPatterns.org community site, and pattern profile tables for design patterns referenced in this book are further provided in Appendix D.

- Definitions for the terms introduced in this chapter can also be found at www.soaglossary.com.
- Read the Annotated SOA Manifesto in Appendix E (also published at www.soa-manifesto.com) for a formal, high level description of SOA and service-orientation.

See www.soabooks.com for additional reading resources, and visit www.soaschool.com and www.cloudschool.com for additional educational resources.

Chapter 4



SOA Planning Fundamentals

4.1 The Four Pillars of Service-Orientation

4.2 Levels of Organizational Maturity

4.3 SOA Funding Models

SOA PRINCIPLES & PATTERNS REFERENCED IN THIS CHAPTER

- Domain Inventory [520]
- Entity Abstraction [524]
- Process Abstraction [544]
- Service Layers [561]
- Utility Abstraction [573]

Governance, as with any other aspect of SOA adoption, begins with the planning stage. This chapter covers some general SOA planning considerations, all of which pertain to the definition of SOA governance strategies but are also themselves affected by the chosen governance approach. We begin with an overview of the foundations required for any SOA initiative, and then move on to describe common evolutionary stages and funding models.

4.1 The Four Pillars of Service-Orientation

As explained in Chapter 3, the attainment of the goals and benefits commonly associated with service-oriented computing and SOA require the application of the service-orientation paradigm. 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 these 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 pre-requisites for its successful adoption. These critical success factors are referred to as *pillars* because they collectively establish a sound and healthy foundation upon which to build, deploy, and govern services.

The four pillars of service-orientation are:

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

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

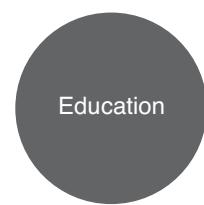
Teamwork

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



Education

A key factor to realizing the reliability and trust required by SOA team members is to ensure that they use a common communications framework based on common vocabulary, definitions, concepts, methods, and a common understanding of the target state the team is collectively working to attain. To achieve a 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 risks that have traditionally plagued SOA projects.

Discipline

A critical success factor for any SOA initiative is consistency in how knowledge and practices amongst 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 scope of SOA adoption can vary. Keep efforts manageable and within meaningful boundaries."

– SOA Manifesto

See Appendix E for the complete SOA Manifesto and the Annotated SOA Manifesto, which are also published at www.soa-manifesto.org and www.soa-manifesto.com, respectively.

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 4.1).

Common factors involved in determining a balanced scope include:

- cultural obstacles
- authority structures
- geography
- business domain alignment
- available stakeholder support and funding
- available IT resources

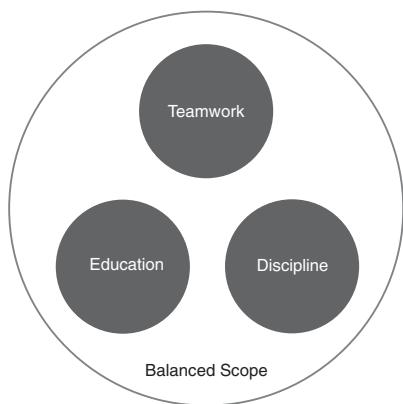


Figure 4.1

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

A single organization can choose one or more balanced adoption scopes (Figure 4.2). Having multiple scopes results in a domain-based approach to adoption. Each domain establishes a boundary for an inventory of services. Among domains, adoption of service-orientation and the delivery of services can occur independently. This does not result in application silos; it establishes meaningful service domains (also known as “continents of services”) within the IT enterprise.

SOA PRINCIPLES & PATTERNS

The domain-based approach to the adoption of SOA and service-orientation originated with the Domain Inventory [520] pattern. However, it is important to acknowledge that logical domains within a domain service inventory can also be established by classifying services based on the nature of their respective functional contexts. This form of separation is relevant to service governance and is the basis of the Service Layers [561] pattern, as well as the related Utility Abstraction [573], Entity Abstraction [524], and Process Abstraction [544] patterns.

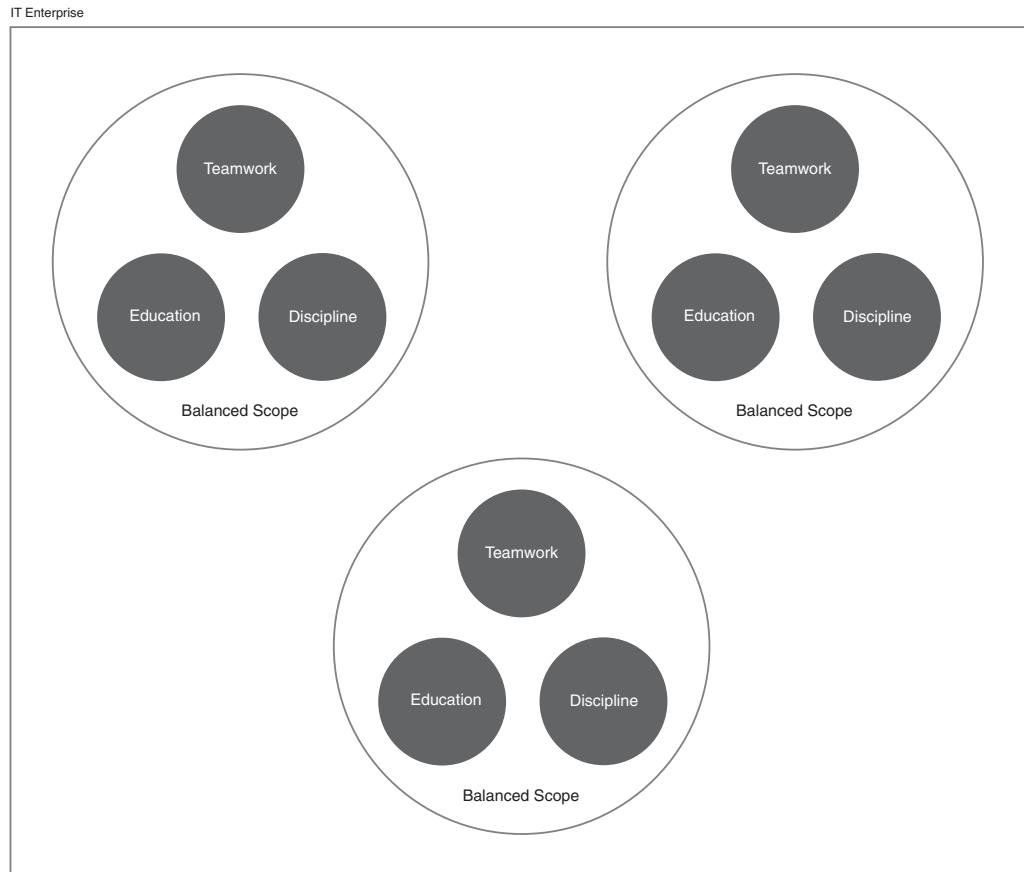


Figure 4.2

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

SUMMARY OF KEY POINTS

- Teamwork, education, and discipline represent foundational critical success factors for the successful adoption of service-orientation.
 - Setting a meaningful and manageable scope of adoption establishes a boundary in which services are to be delivered and consequently determines the extent to which these three critical success factors need to be realized.
 - Setting a balanced scope is a strategic planning decision and therefore itself a critical success factor.
-

4.2 Levels of Organizational Maturity

From the point at which an organization begins planning for the adoption of SOA and service-orientation up until the time it achieves its planned target state, it can transition through one or more of the following common evolutionary levels:

- Service Neutral
- Service Aware
- Service Capable
- Business Aligned
- Business Driven

Each of these levels represents a state of maturity of an organization on its way to carrying out a legitimate SOA adoption project based on the proper foundations. However, additional levels are worth noting for when organizations proceed with SOA initiatives in the absence of some or all of the previously described four pillars:

- Service Ineffectual
- Service Aggressive

Figure 4.3 displays how an organization will commonly transition through (positive and negative) maturity levels.

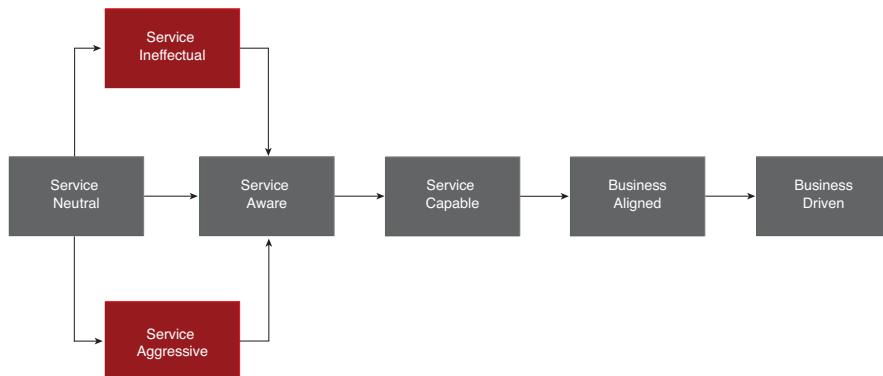


Figure 4.3

Common evolutionary levels of organizational maturity.

Service Neutral Level

This level indicates that there may be an awareness of SOA and service-orientation within the organization, but no meaningful extent of teamwork, education, or discipline has been established or yet identified. Every organization begins at the Service Neutral level, as it represents the starting point in the evolutionary lifecycle. Whereas the Service Neutral level indicates an absence of maturity because the organization has not yet proceeded with an adoption effort, the Service Ineffectual and Service Aggressive levels represent an absence of maturity during the adoption effort. Figure 4.3 illustrates this by positioning this level at the beginning of the lifecycle. From this point, an organization can either move on to the Service Ineffectual or Service Aggressive levels, or it can proceed to the Service Aware level.

Service
Neutral

Service Aware Level

When reaching the Service Aware level, it has been confirmed that the four pillars have been established, that relevant business requirements and goals are defined, and that the overall necessary organizational foundation for the SOA initiative is in place. Within this context, the term “service aware” does not refer to an IT enterprise becoming aware of SOA and service-orientation; it refers to an early planning stage that validates that the necessary foundations (pillars) and business direction for a planned SOA initiative are identified and defined.

Service
Aware

Service Capable Level

When the organization achieves the ability to deliver and govern services and service compositions in response to business automation requirements, it has reached the Service Capable level. An organization at this level will have avoided or overcome common adoption pitfalls and will therefore be positioned for a successful adoption. It will have a skilled, well trained team that has consistently delivered services within the required processes and regulations.

Service
Capable

The principal risk at this level is that of stalling—remaining at a Service Capable level, without making further progress to move on to the Business Aligned level. This risk exists due to a potential sense of satisfaction with the current state of having services capable of composition.

Business Aligned Level

This level indicates that the organization has achieved meaningful alignment of (service-oriented) technology resources and current business automation requirements. In other words, the organization has successfully aligned services and service compositions with the current state of the business.

Business
Aligned

Attaining this state further implies that most or all of the services planned for a given service inventory have been delivered and are in operation. Therefore, the Business Aligned level represents a level of organizational maturity that has resulted from having established a relatively mature service inventory.

Business Driven Level

This evolutionary level represents a state where service-encapsulated technology resources are not just aligned with the current state of the business, but have proven themselves able to remain in alignment with how business requirements continue to change. This form of evolutionary alignment is accomplished via the repeated or augmented composition of services. This level therefore represents the highest level of maturity for a service inventory as well as the highest level of success for the overall SOA adoption effort. The Business Driven level can last indefinitely as the organization continues to leverage the strategic benefits of its services.

Business
Driven

NOTE

The attainment of this level is commonly associated with the successful utilization of an SOA governance vitality framework, as explained in Chapter 13.

Service Ineffectual Level

The Service Ineffectual level occurs when an organization descends into a technological backwater where the IT enterprise delivers services as silo-based or bottom-up automation solutions under the pretense that it is adopting SOA. Services delivered during this level are generally not actual units of service-oriented logic. They are most likely single-purpose software programs labeled as services because they use one or more forms of service

Service
Ineffectual

technology (such as technologies associated with Web services, REST service, and cloud platforms).

This level represents an IT initiative that, under the guise of “SOA,” is tactically focused without much regard for service-orientation or the steps necessary to attain the strategic target state associated with SOA and service-oriented computing.

NOTE

So many IT projects have fallen victim to this pitfall that it has tarnished the perception of “SOA” in general, leading to the need for the SOA Manifesto to be declared in 2009.

Service Aggressive Level

When an organization is Service Aggressive, it is usually because IT’s enthusiasm for SOA and service technology has led to a proliferation of services that the business doesn’t want or need; in some cases, the business may not even be aware of their existence. The Service Aggressive level is different from the Service Ineffectual level in that there may be a sincere intention to adopt SOA and service-orientation in support of strategic goals. However, due to lack of teamwork or education or discipline or perhaps due to blatant incompetence, the SOA initiative fails to align its technology in support of the business. This misalignment therefore severely limits the usefulness and longevity of delivered services.

Service
Aggressive

SUMMARY OF KEY POINTS

- There are common levels an organization can transition through when adopting SOA and service-orientation.
 - The Service Neutral level is the typical starting point for an organization. From this level, it may transition to the Service Aware, Service Ineffectual, or Service Aggressive level.
 - If an organization successfully transitions to the Service Aware level, it can move on to the Service Capable, Business Aligned, and Business Driven levels, each of which are supportive of achieving desired strategic goals.
 - The Service Ineffectual and Service Aggressive levels are considered anti-patterns and inhibitors of successful SOA adoption.
-

4.3 SOA Funding Models

No IT project can succeed or proceed without funding. Traditional projects focused on the delivery of silo-based applications typically had straightforward funding models. Because of the single-purpose nature of the application logic, it was relatively simple to calculate the costs of delivery and the anticipated return on investment. In fact, this predictability factor is one of the strengths of silo-based delivery approaches.

With SOA projects, funding is more complex due to the strategic goals we aim to achieve and work toward with the delivery of each service. Funding becomes the concern of those delivering the services, those expecting to reuse the services, and those responsible for establishing an environment for multiple services. Furthermore, being able to show the returns on funding investments becomes another primary area of interest to those providing the funding and to those using the funds.

Therefore, establishing proper funding mechanisms and a means of proving consistent benefits and returns are critical to the success of any SOA governance program.

In this section, we explore two fundamental levels of SOA funding:

- *Platform Funding* – This level provides funds for the delivery of collections or inventories of services. Platform funding models are therefore generally focused on establishing the supporting infrastructure for a given service inventory.
- *Service Funding* – This level provides funds for the delivery of individual services. These services rely on an already-funded service inventory platform.

Furthermore, we will discuss how to capture and communicate the benefits and ROI of the platforms and services that have been funded. Once these models are in place and the necessary financial elements are identified, tangible and implied benefits can be calculated.

Platform (Service Inventory) Funding

This level of funding is focused on service inventory architecture and infrastructure. The intent is to establish an environment capable of hosting and enabling the runtime utilization of a collection of related services. One of the key factors here is that services need to be repeatedly leveraged as part of new and reconfigured service compositions.

Generally, the emphasis is on infrastructure components, such as service registries, ESBs, and various forms of service management frameworks. The important distinction is that this funding is not for the actual definition or creation of services.

There are three common platform funding models:

- *Project Funding Model* – This model identifies appropriate projects to acquire or to extend new or existing SOA platforms.
- *Central Funding Model* – With this model, the funding of the platform is provided centrally.
- *Usage Based Funding Model* – The initial platform is funded centrally but usage fees are charged for ongoing maintenance and support.

Let's look at each model individually:

Project Funding Model (Platform)

This model calls for attaching platform costs to individual service delivery projects (Figures 4.4 and 4.5). With this approach, projects are required to purchase and implement (or upgrade) any new infrastructure components necessary to automate their planned services. These implementations are limited to the pre-defined scope of the service inventory architecture. The project funding model enables a service inventory architecture to grow organically, based on individual project requirements.

Pros

- fits well with traditional, silo-based funding models

Cons

- high potential for conflicts with other projects that need to use platform resources or services funded by initial projects
- platform costs may inflate project budget
- project scope may not address service inventory-wide needs
- may result in unclear support and ownership requirements
- standardization issues may arise when different projects build upon a common platform (especially with cloud-based environments)

Although the simplicity of this model can be beneficial, it has several drawbacks. Despite this, there may be circumstances where a project funding model is the only option supported by stakeholders, primarily due to the familiarity of this model with traditional application development projects—or—simply due to a lack of sponsor support for the other funding models.

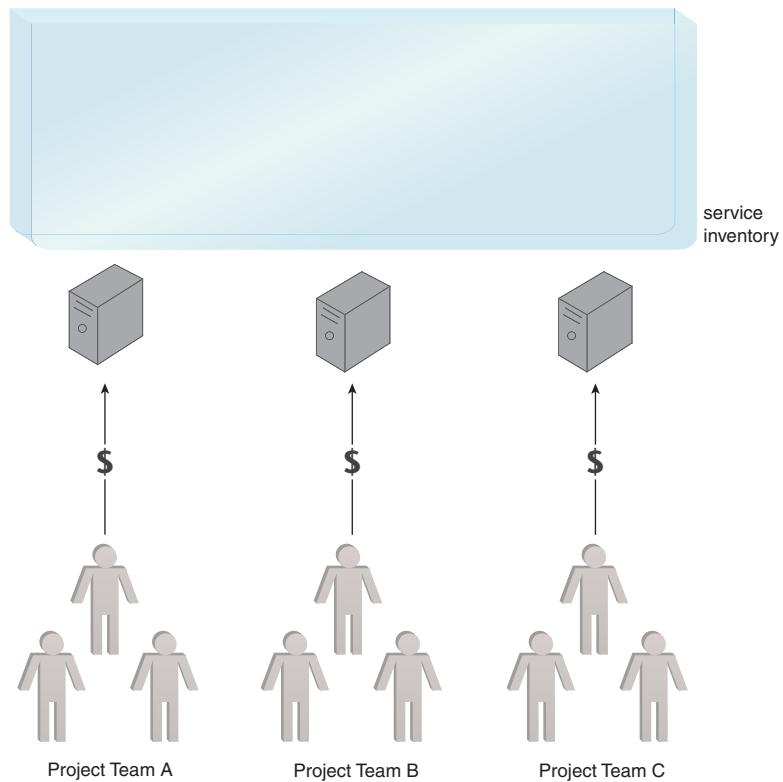


Figure 4.4

On-premise infrastructure resources for the service inventory is gradually (and often iteratively) assembled by individual service delivery projects. Shown are the physical servers being funded by individual project teams.

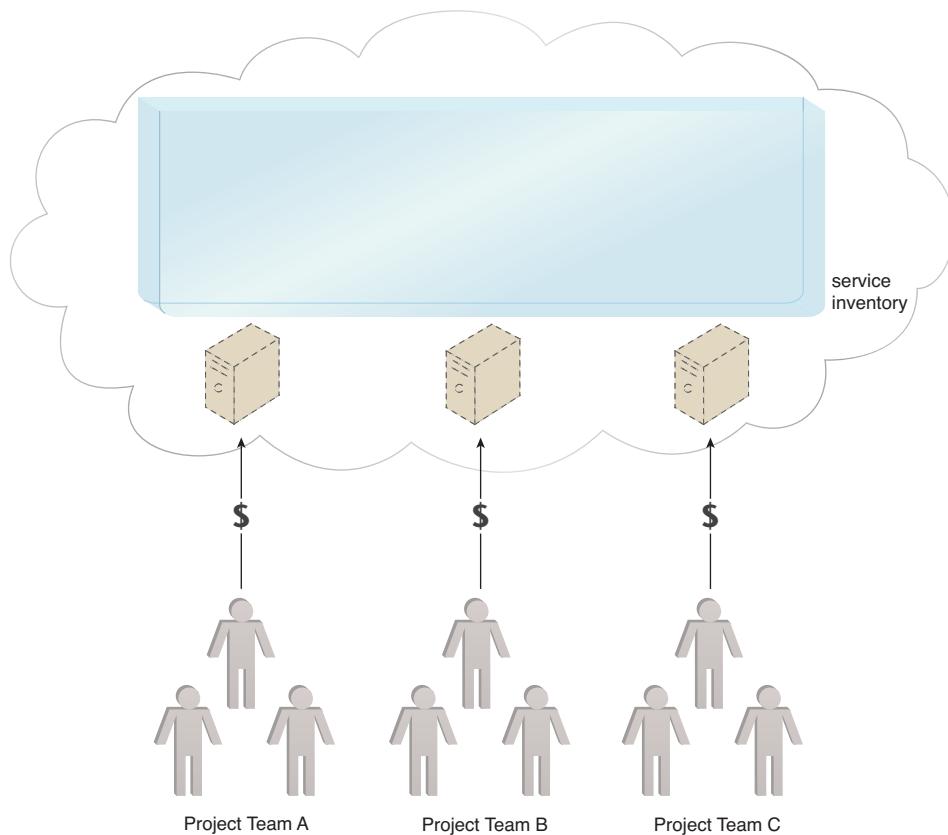


Figure 4.5

Cloud-based infrastructure resources are leased, although up-front set-up costs and leasing terms can still require advance funding. Shown are virtual servers being leased by individual project teams.

Central Funding Model (Platform)

The Central Funding Model entails funding all of the SOA platform build-out and growth activities centrally (Figures 4.6 and 4.7), as they apply to a given service inventory. The central funding source is established and used for new acquisitions, expansion of existing platforms, and related maintenance and support efforts. This also includes any resources or labor required to effectively manage the platform and adjust it to accommodate the needs of new services and solution requirements.

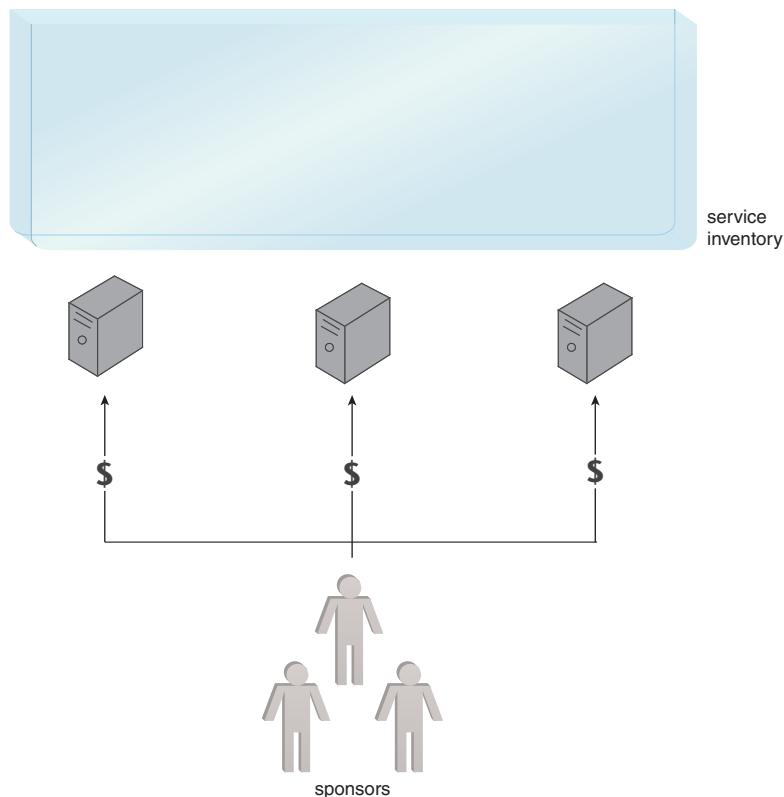


Figure 4.6

The SOA platform evolution is guided by the SOA roadmap under the Central Funding Model and the necessary on-site platform infrastructure is funded by one group of sponsors.

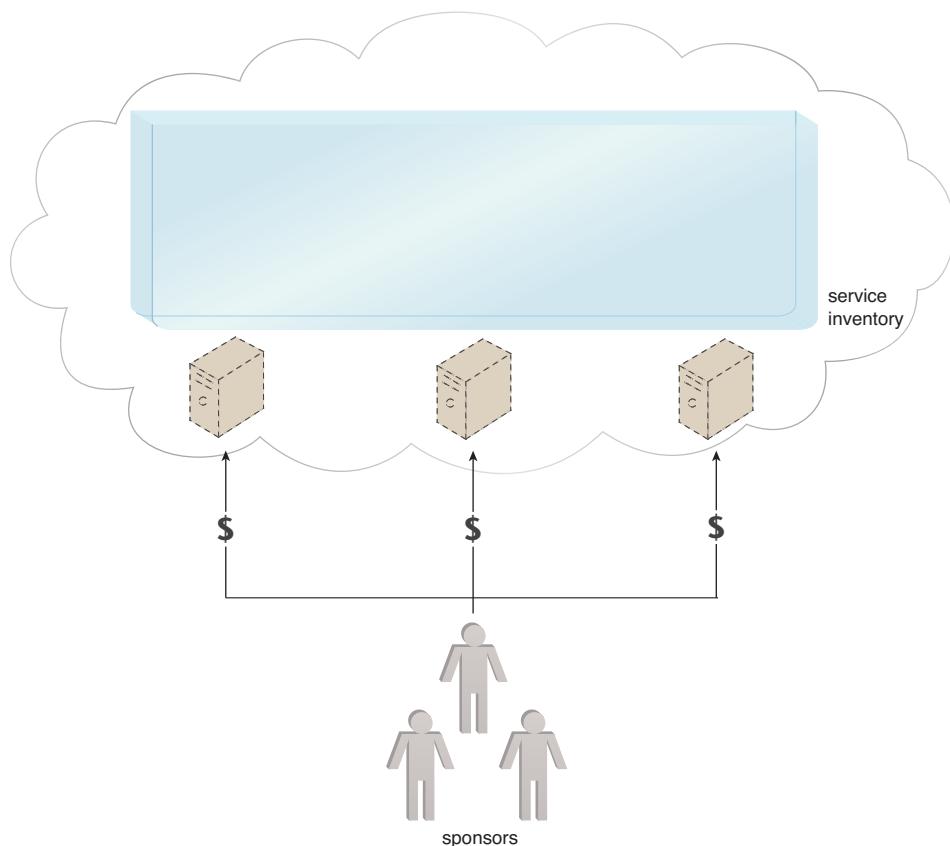


Figure 4.7

This model can also be applied to cloud-based infrastructure resources, whereby a central group of sponsors establishes a business relationship with a cloud provider and funds costs as they are incurred.

Central funding allows an organization to establish an independent roadmap for introducing and upgrading SOA platforms because cost, scaling, and availability issues are no longer relevant to individual projects.

Pros

- covers the platform needs of an entire service inventory with no additional funding sources required
- high visibility into platform expansion, scalability, and usage needs
- highly predictable support and ownership model
- funding occurs independent of individual project demands

Cons

- it can be difficult to secure consistent and ongoing enterprise level funding
- more accountability for spending and budgets may be required due to increased scope and responsibility

This model places increased emphasis on careful and diligent management in order to continually demonstrate how and where the funds went, while continuing to provide the necessary platform and infrastructure support for a growing service inventory.

Usage Based Funding Model (Platform)

This approach can be considered an extension of the Central Funding Model in that the platform builds out and expansion is still funded centrally; however, services and solutions that make use of platform resources are charged fees in order for investments to be recouped (Figures 4.8 and 4.9). The Usage Based Funding Model is especially commonplace when sharing resources via cloud environments. Depending on whether a public or private cloud is being used, usage fees may be charged by a central department within the organization or a third-party cloud provider.

Regardless of whether a cloud environment is being utilized, common types of fees can include the following:

- *Entry Fees* – These are standard one-time charges for new projects. These charges may be uniform across each service inventory platform or determined independently. Entry fees are typically calculated based on the general effort required to setup and support a new project on the platform, plus the prorated cost of the platform's portion being utilized.
- *Per Use Fees* – These charges are calculated using formulas based on unit costs and total fees for a predefined period of time. Total charges are based on the amount of units used within an invoiced period.
- *Supplemental Fees* – The more a service utilizes IT resources within the cloud, the more additional “supplemental” fees may be required to cover its build-out, maintenance, support, and growth.

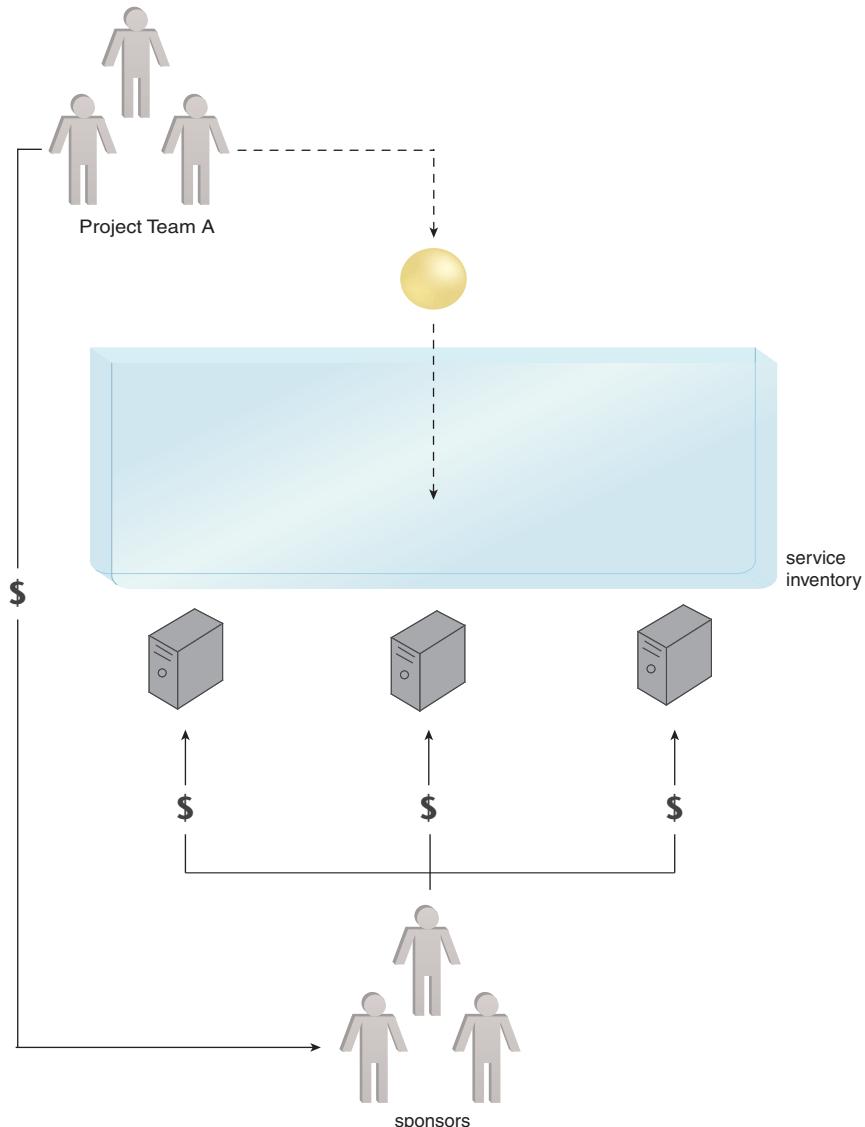


Figure 4.8

Under the Usage Based Funding Model, the SOA platform is still funded by a group of sponsors, but the on-site investment is recouped through usage fees.

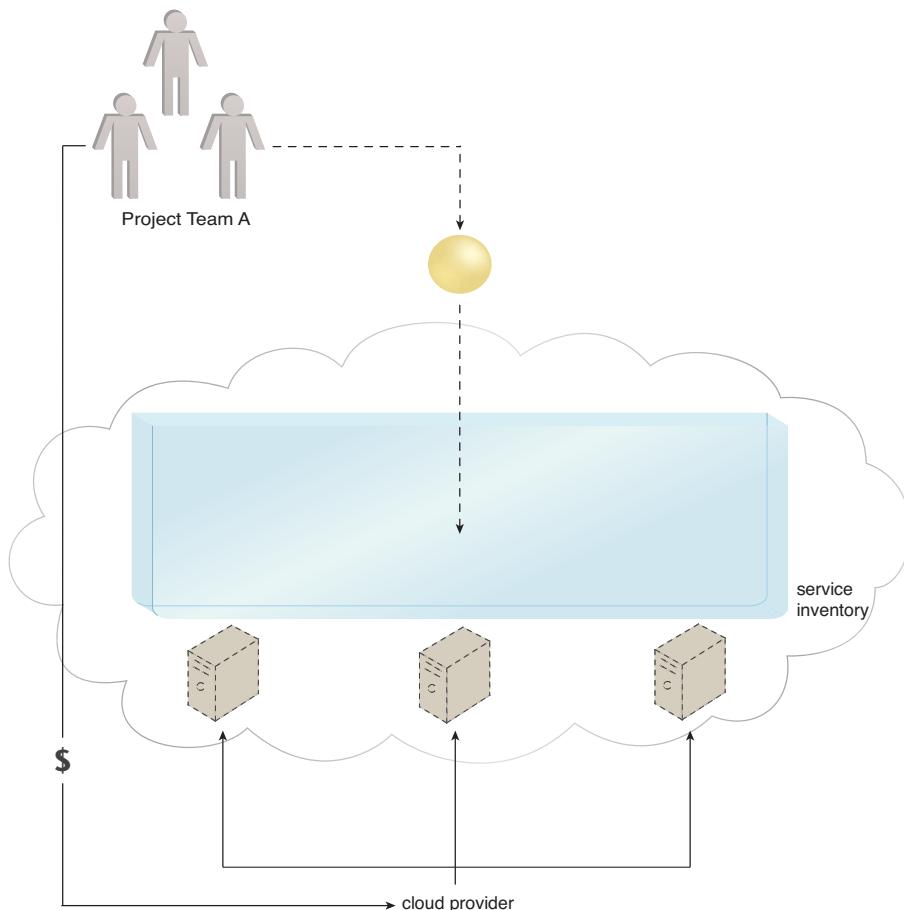


Figure 4.9

A variation of this model is comparable with the leasing arrangement of a public cloud provider, whereby the cloud provider simply charges the project team for infrastructure resources on a per-usage basis.

The following pros and cons apply to all forms of usage fees. Note that the pros and cons listed under the Central Funding Model generally also apply to the Usage Based Funding Model.

Pros

- can rapidly recoup the investment made in the platform
- charges are proportional to platform usage
- return on investment can be more easily demonstrated

Cons

- efficient and trusted chargeback mechanisms needed to monitor and bill for usage
- recovery of funds requires new billing processes
- improperly priced usage charges can deter projects from using platform resources

Service Funding

Agnostic services are created with the expectation that they will be reused and will provide repeated return on the investment originally required for their delivery. Because they are positioned as shared services, they are not dedicated to any one project. Instead, they are to be used to automate different business processes for which different projects build different solutions. Therefore, the funding of these reusable services requires special attention.

There are several different funding models that can be utilized:

- *Project Funding Model* – Individual projects build new or extend shared existing services.
- *Central Funding Model* – The funding of shared services is provided centrally.
- *Usage Based Funding Model* – The initial delivery of shared services is funded centrally but usage fees are subsequently charged.
- *Hybrid Funding Model* – A combination of project and central funding.

Let's look at each model individually:

Project Funding Model (Service)

With this model, a project responsible for delivering a solution for the automation of a specific business process is required to identify and build reusable services (Figure 4.10). Although it may seem a simple and familiar approach for organizations accustomed to silo-based application delivery, this model places the burden of creating (and possibly maintaining) services that may be used throughout the enterprise upon a single project. The project may be given additional funding, but it can be difficult for project team members, for whom the priority is the automation of a particular business process, to ensure that shared services are designed as truly reusable and re-composable enterprise resources.

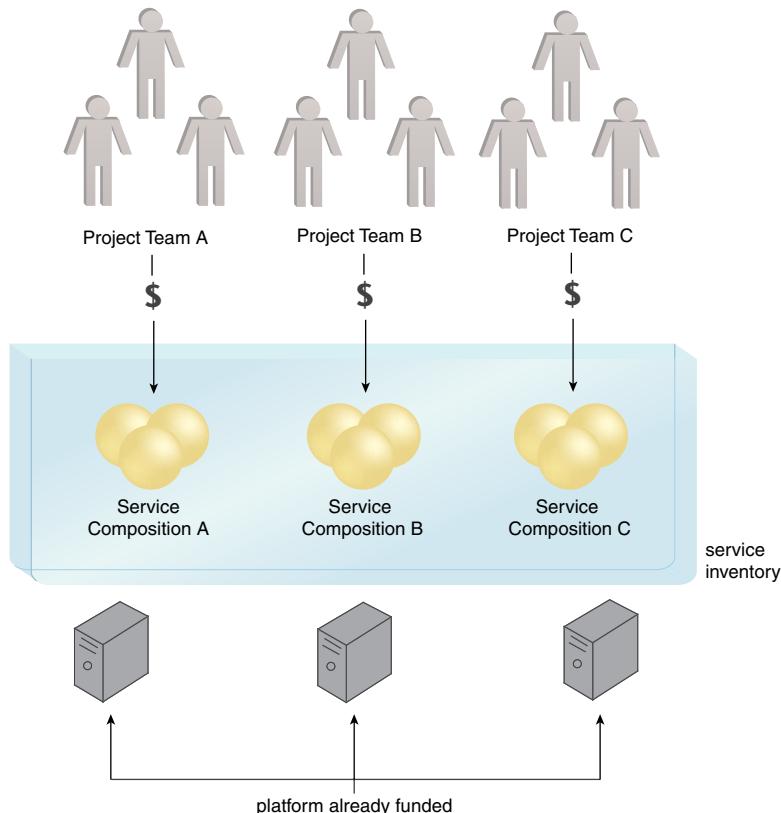


Figure 4.10

The service inventory grows based on project demand under the Project Funding Model. (The same funding mechanism applies if services are deployed in a cloud.)

Pros

- more compatible with familiar silo-based delivery project models
- increased efficiency in the delivery of reusable services

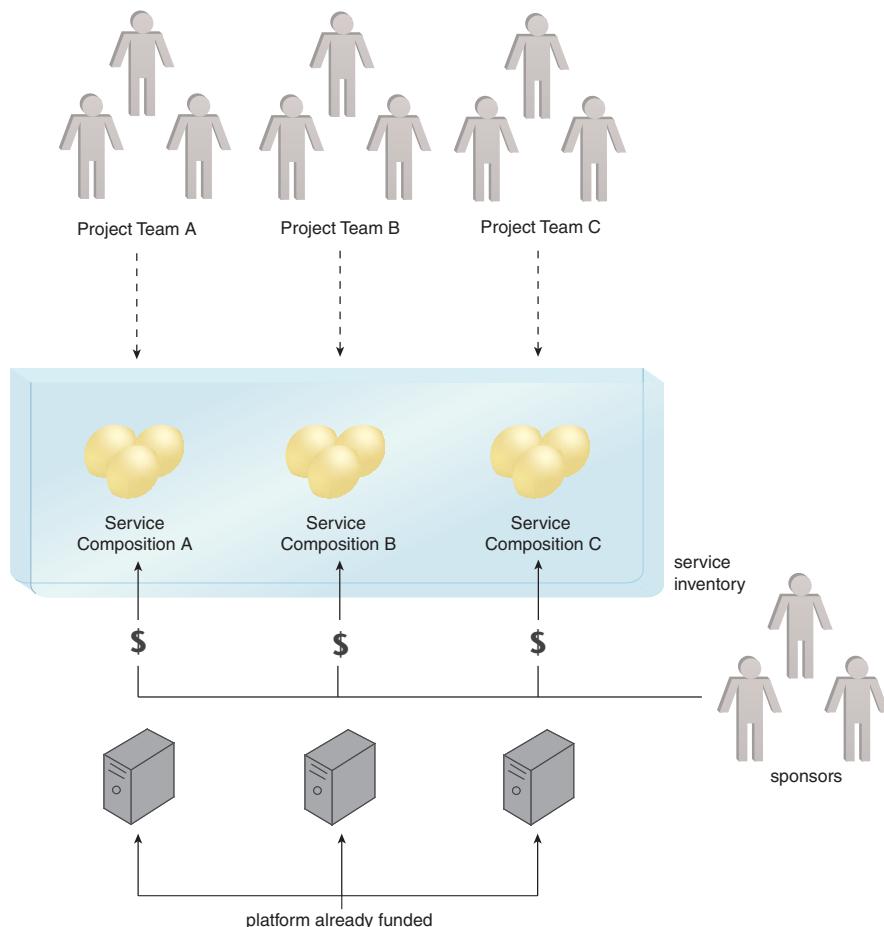
Cons

- high risk of producing low-quality reusable services
- unclear support and ownership model
- can be difficult to measure the success of the project

Central Funding Model (Service)

Within the Central Funding Model the funding for building and maintaining shared and reusable services within the same service inventory comes from a sole pool of funds (Figure 4.11). As a result, individual project budgets are only impacted when:

- a new reusable service needs to be built (or an existing reusable service needs to be extended) and the project needs to wait for the central department to perform the development effort
- an existing reusable service needs to be used and there is cost and effort associated with incorporating it into the new solution

**Figure 4.11**

Under the Central Funding Model, the service inventory grows based on project demand but is funded by a central group of sponsors. (The same funding mechanism applies if services are deployed in a cloud.)

This model is common with organizations that have also chosen the Central Funding Model for platform funding purposes. In this case, the source of both forms of funding is likely the same.

Pros

- establishes a clear support and ownership model
- can help accelerate service reuse
- simplifies service governance

Cons

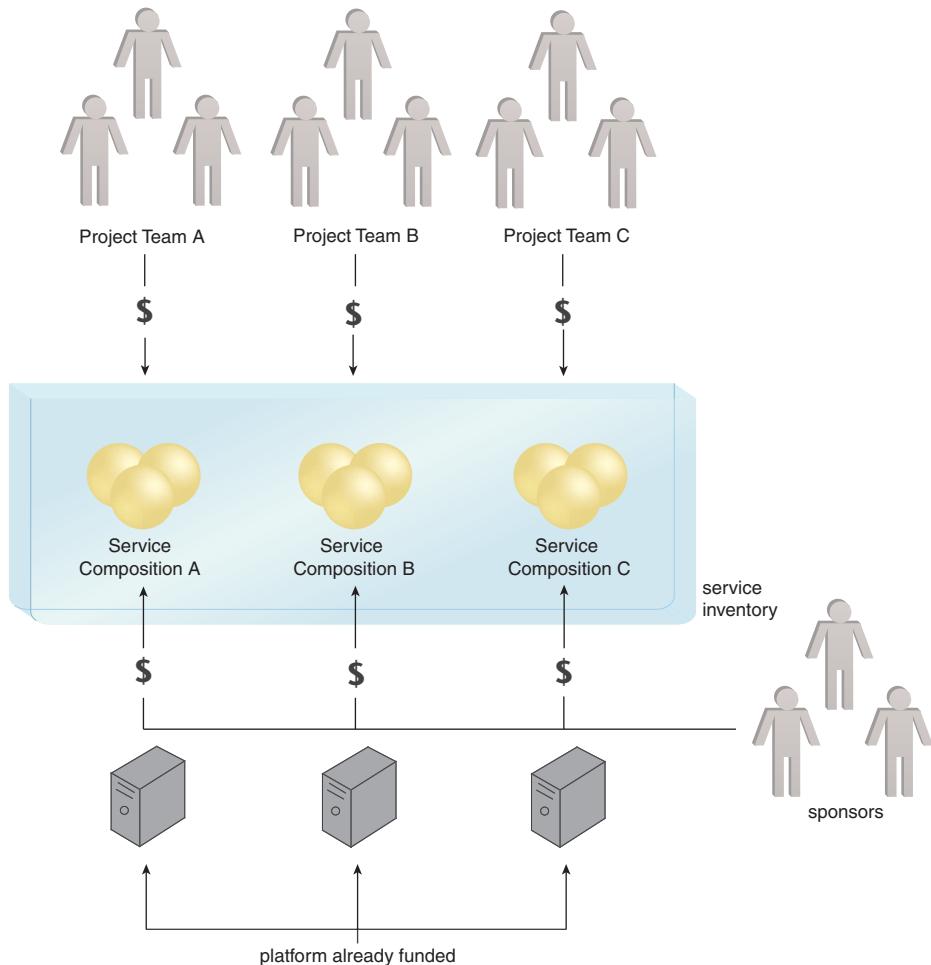
- can increase opportunity for wasteful spending or “gold-plating” of reusable services
- can introduce scalability challenges

Hybrid Funding Model (Service)

This model is based on a mixed funding framework where some funding comes from project budgets, while the rest is supplied by a central source. In other words, it's a hybrid of the Project and Central Funding Models (Figure 4.12). The intent is to increase the ability to deliver truly reusable services based on project demand without over-taxing the individual project budgets.

A central fund needs to be established to cover the efforts falling outside of each project scope. Since shared services typically incorporate other projects as well as additional enterprise requirements in order to meet reusability goals, the actual cost ends up being higher than what projects have budgeted for their needs. Therefore, supplementary funding is distributed to allow projects to pay for functionality already included in their budgets, and to then cover the additional costs through a central fund.

The Hybrid Funding Model attempts to balance between project demands and enterprise goals. With this approach, projects no longer have to be concerned about additional funds required to make services truly reusable, injecting additional enterprise requirements not relevant to their immediate solution, or accepting work from other teams. The central funds take care of these requirements and who performs the actual work also becomes largely irrelevant.

**Figure 4.12**

Under the Hybrid Funding Model, the service inventory grows based on project demand while the project funding is supplemented from the central SOA fund. (The same funding mechanism applies if services are deployed in a cloud.)

Pros

- can balance project (tactical) and enterprise (strategic) goals
- can eliminate problems associated with the Project and Central Funding models

Cons

- can be hard to establish and maintain throughout the growth of a service inventory
- creates opportunities for abuse by projects
- may introduce complex service support and ownership requirements

Usage Based Funding Model (Service)

The Usage Based Funding Model for services is similar to the platform funding model of the same name, in that both are based on central funding that then requires usage fees to be charged to consumers (Figure 4.13). Usage-based services hosted in cloud platforms are comparable to the Software-as-a-Service (SaaS) cloud delivery model, especially when offered by third-party public cloud providers (Figure 4.14).

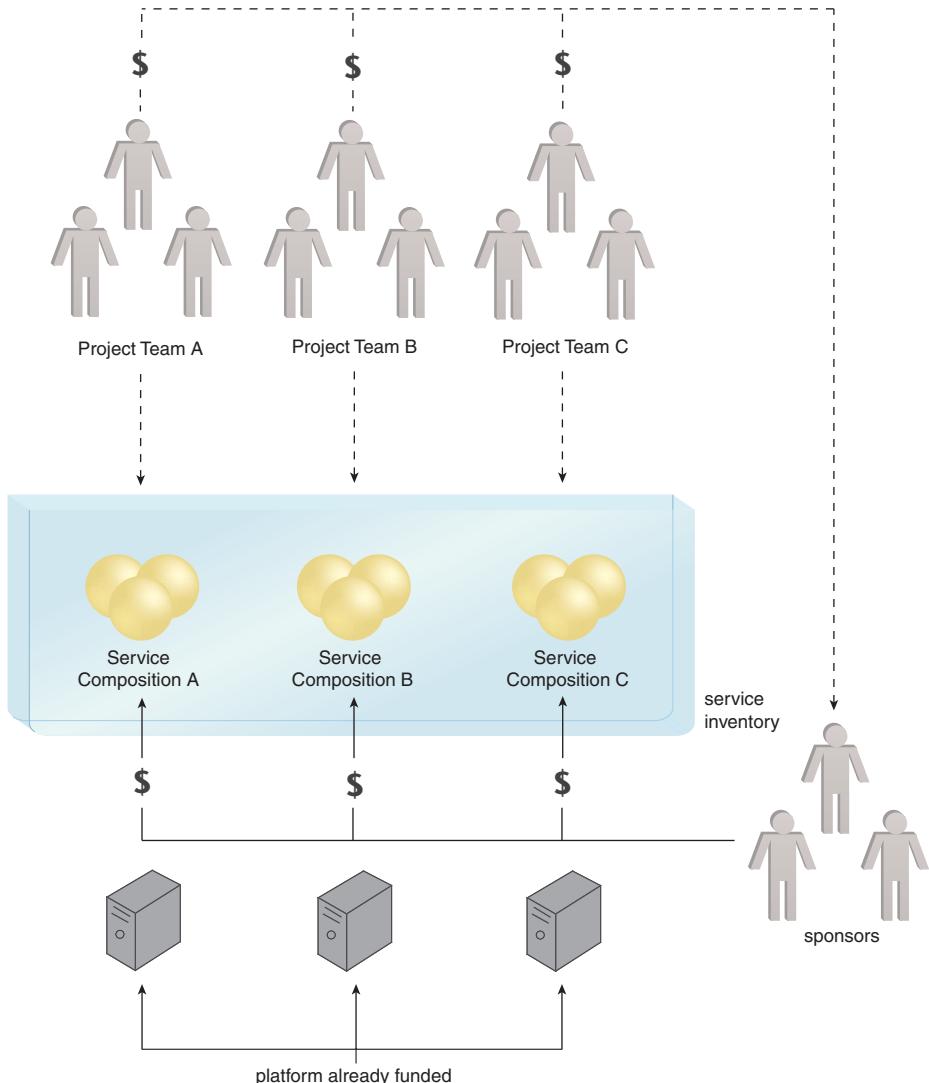
The pricing model is generally based on a system of charging nominal fees for each instance of a service used by a given service consumer. Furthermore, an entry fee may be charged for the first time a new solution requests access to the service. Over time, the initial investment of the service is recouped and then, hopefully, far succeeded with the collected funds.

Pros

- defines a clear system for investment recovery
- eliminates the need for additional forms of non-project funding
- stream of funds help empower Service Custodians to evolve and improve reusable services

Cons

- depends on the usage of billing mechanisms
- can be difficult to set appropriate fee structures that accommodate and are fair to all potential service consumers

**Figure 4.13**

Under the Usage Based Funding Model, the sponsors that centrally fund the service delivery charge project teams for the usage of the services.

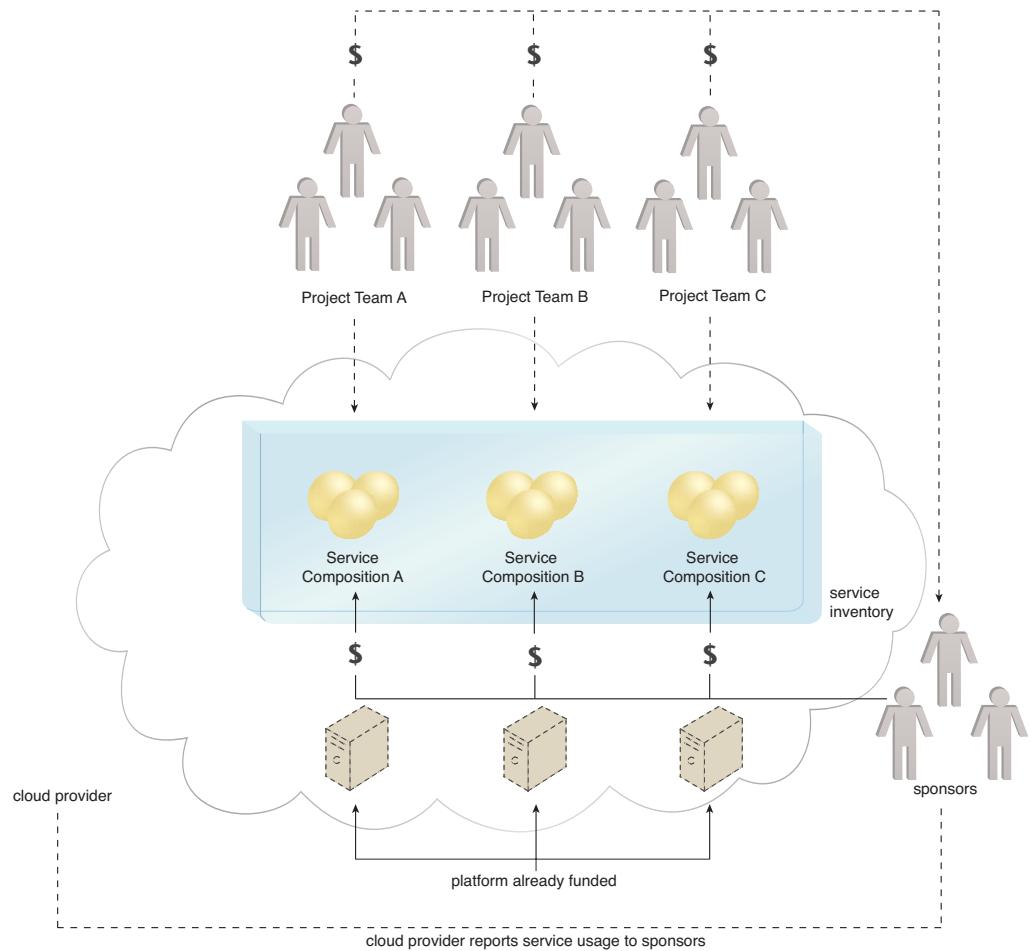


Figure 4.14

When the Central Funding Model is applied to a cloud-based service inventory, the sponsors can still fund the service delivery, but they can receive actual usage costs to bill the project teams via pay-for-use monitoring and reporting from the cloud provider.

SUMMARY OF KEY POINTS

- Funding is generally planned on two levels: platform (service inventory) and service.
 - Project-specific funding models can be easier to implement but can result in problems that inhibit successful SOA adoption.
 - Central funding models are more conducive to achieving SOA goals, but can be more challenging to establish (especially in organizations unaccustomed to shared funding and reusable resources).
 - Other funding models can exist, some of which are comprised of a combination of common models.
-

This page intentionally left blank

Chapter 5



SOA Project Fundamentals

5.1 Project and Lifecycle Stages

5.2 Organizational Roles

5.3 Service Profiles

This book does not comprehensively define service project lifecycle stages or organizational roles associated with SOA projects. These areas, for the most part, have already been well-documented in previous titles in this book series. As an introductory resource, this chapter provides a set of sections comprised of summarized content from the following books and courses, along with new content introduced in support of upcoming chapters:

- *Service-Oriented Architecture: Concepts, Technology, and Design* – Contains content about SOA project delivery processes and roles.
- *SOA Principles of Service Design* – Contains additional content about SOA project roles, as well as the description of service profiles.
- *Web Service Contract Design and Versioning for SOA* – Contains detailed content about service contract versioning.
- *Module 4: SOA Project Delivery & Methodology* – Contains definitions for project stages and roles in relation to different methodologies (from the *SOA Certified Professional* curriculum).
- *Module 1: Cloud Computing Fundamentals* – Contains definitions for organizational roles and concepts specific to cloud computing (from the *Cloud Certified Professional* curriculum).

If you are new to service project lifecycle stages, organizational roles, and the use of service profiles, then it is recommended that you study the upcoming sections as these topics will be referenced throughout upcoming chapters.

NOTE

Several of the upcoming sections and sub-sections contain figures that do not have corresponding inline text references. This is intentional so as to improve the flow of content. The figures in question are displayed in close proximity to their sections, making the association evident.

5.1 Project and Lifecycle Stages

The following represent common and primary stages (or phases) related to SOA projects and the overall service lifecycle:

- SOA Adoption Planning
- Service Inventory Analysis
- Service-Oriented Analysis (Service Modeling)
- Service-Oriented Design (Service Contract)
- Service Logic Design
- Service Development
- Service Testing
- Service Deployment and Maintenance
- Service Usage and Monitoring
- Service Discovery
- Service Versioning and Retirement

Although Figure 5.1 displays stages sequentially, how and when each stage is carried out depends on the methodology being used. Different methodologies can be considered, depending on the nature and scope of the overall SOA project, the size and extent of standardization of the service inventory for which services are being delivered, and the

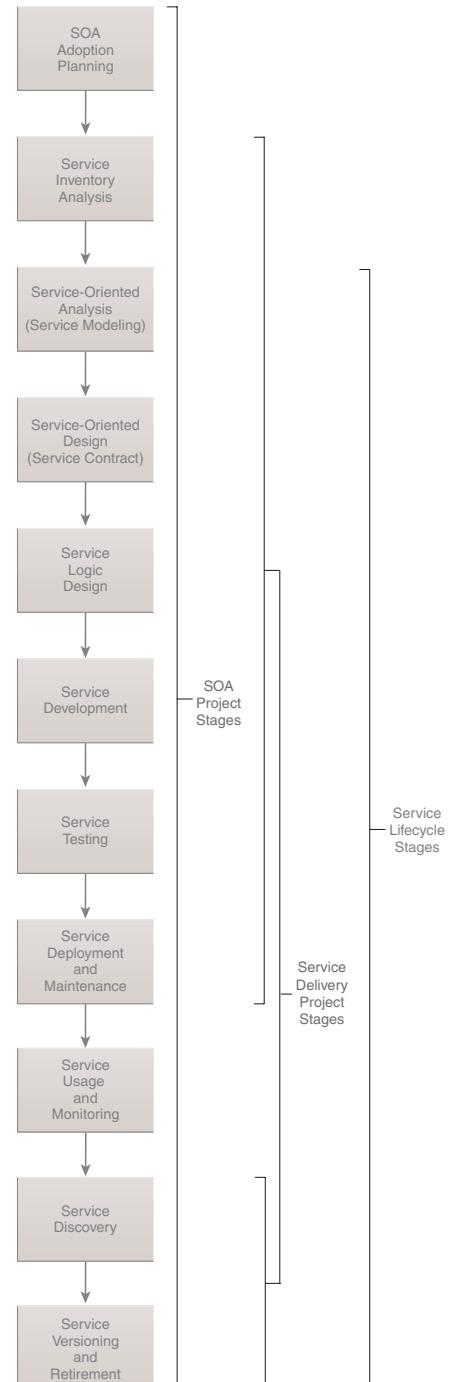


Figure 5.1

Common stages associated with SOA projects. Note the distinction between SOA project stages, service delivery project stages, and service lifecycle stages. These terms are used in subsequent chapters when referring to the overall adoption project, the delivery of individual services, and service-specific lifecycle issues, respectively.

manner in which tactical (short-term) requirements are being prioritized in relation to strategic (long-term) requirements.

A fundamental characteristic of SOA projects is that they tend to emphasize the need for some meaningful extent of strategic target state that the delivery of each service is intended to support. In order to realize this, some level of increased up-front analysis effort is generally necessary. Therefore, a primary way in which SOA project delivery methodologies differ is in how they position and prioritize analysis-related phases.

The upcoming sections briefly describe all stages:

SOA Adoption Planning

It is during this initial stage that foundational planning decisions are made. These decisions will shape the entire project, which is why this is considered a critical stage that may require separately allocated funding and time in order to carry out the studies required to assess and determine a range of factors, including:

- scope of planned service inventory and the ultimate target state
- milestones representing intermediate target states
- timeline for the completion of milestones and the overall adoption effort
- available funding and suitable funding model
- governance system
- management system
- methodology
- risk assessment

Additionally, prerequisite requirements need to be defined in order to establish criteria used to determine the overall viability of the SOA adoption. The basis of these requirements typically originates with the four pillars of service-orientation described earlier in Chapter 4.

Service Inventory Analysis

As explained in Chapter 3, a service inventory represents a collection of independently standardized, owned, and governed services. The scope of a service inventory is expected to be meaningfully “cross-silo,” which generally implies that it encompasses multiple business processes or operational areas within an organization.

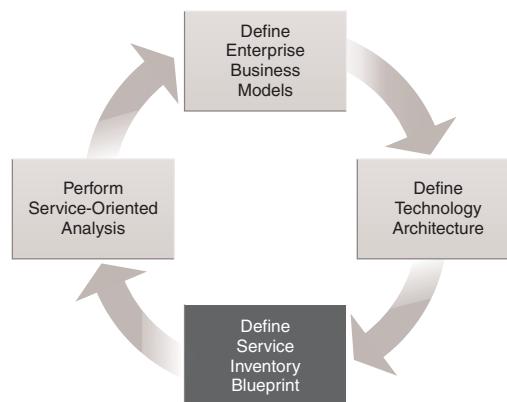
This stage is dedicated to conceptually defining the service inventory so that individual service candidates can be identified and assigned appropriate functional contexts in relation to each other. This ensures that services (within the service inventory boundary) are normalized in that they don't overlap. As a result, service reuse is maximized and the separation of concerns is cleanly carried out. A primary deliverable produced during this stage is the service inventory blueprint.

The scope of the initiative and the size of the target service inventory tend to determine the amount of up-front effort required to create a complete service inventory blueprint. More up-front analysis results in a better defined conceptual blueprint, which is intended to lead to the creation of a better quality inventory of services. Less up-front analysis leads to partial or less well-defined service inventory blueprints.

The Service Inventory Analysis stage is commonly carried out iteratively as part of the Service Inventory Analysis cycle (Figure 5.2). This is comprised of an iterative cycle during which the service inventory blueprint is incrementally defined as a result of repeated iterations of steps that include the Service-Oriented Analysis.

Figure 5.2

The Service Inventory Analysis cycle.



NOTE

The scope of the Service Inventory Analysis stage and the resulting service inventory blueprint directly relates to the Balanced Scope consideration explained in Chapter 4 as part of the section *The Four Pillars of Service-Orientation*.

The enterprise business models referred to in the *Define Enterprise Business Models* step of the Service Inventory Analysis cycle relate to a number of the artifacts (and associated precepts and processes) covered in Chapter 12.

Service-Oriented Analysis (Service Modeling)

Service-Oriented Analysis represents one of the early stages in an SOA initiative and the first phase in the service delivery cycle. It is a process that begins with preparatory information gathering steps that are completed in support of a service modeling subprocess that results in the creation of conceptual service candidates, service capability candidates, and service composition candidates (Figures 5.3 and 5.4).

The Service-Oriented Analysis process is commonly carried out iteratively, once for each business process. Typically, the delivery of a service inventory determines a scope that represents a meaningful domain of the enterprise, or the enterprise as a whole. All iterations of a Service-Oriented Analysis then pertain to that scope, with each iteration contributing to the service inventory blueprint.

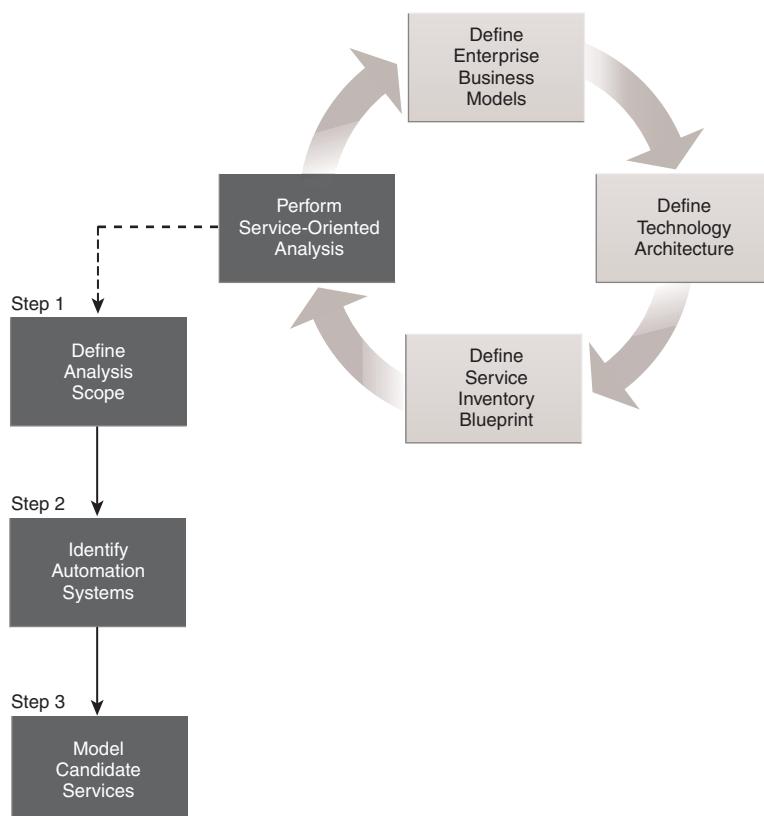
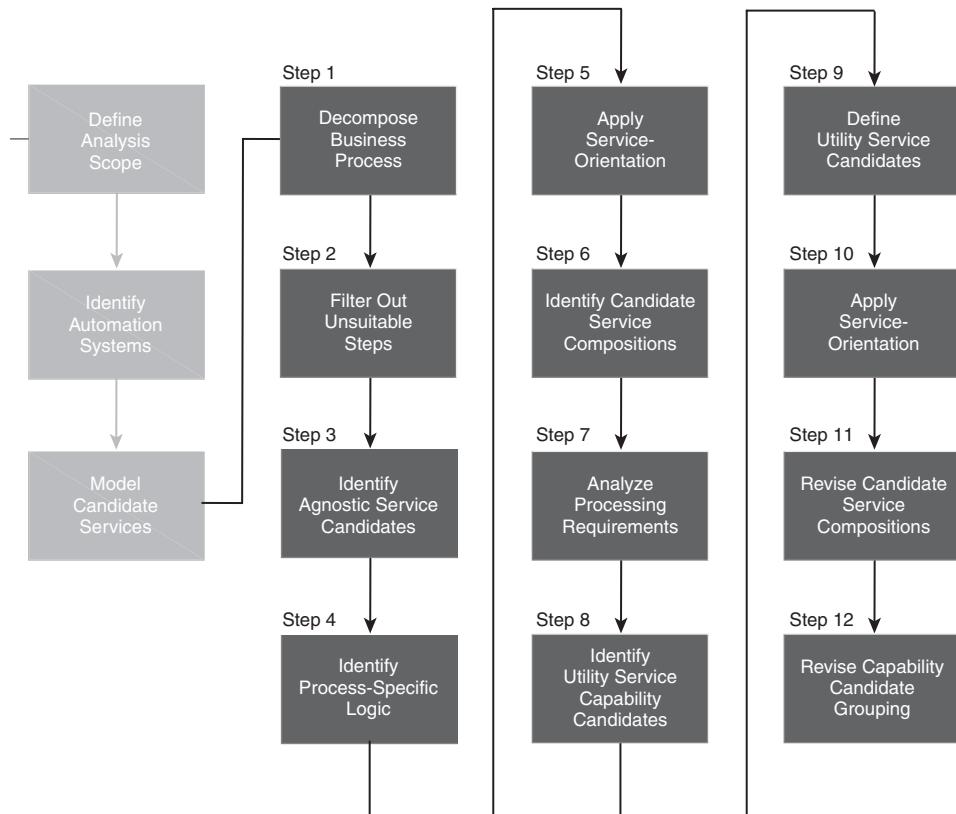


Figure 5.3

A generic Service-Oriented Analysis process that can be further customized. The first two steps essentially collect information in preparation for a detailed service modeling sub-process (Step 3).

**Figure 5.4**

A generic service modeling process comprised of steps that raise key service definition considerations.

A key success factor of the Service-Oriented Analysis process is the hands-on collaboration of both Business Analysts and Technology Architects. The former group is especially involved in the definition of service candidates with a business-centric functional context because they best understand the business logic used as input for the analysis.

Service-Oriented Design (Service Contract)

The Service-Oriented Design phase represents a service delivery lifecycle stage dedicated to producing service contracts in support of the well-established “contract-first” approach to software development.

The typical starting point for the Service-Oriented Design process is a service candidate that was produced as a result of completing all required iterations of the Service-Oriented Analysis process (Figure 5.5). Service-Oriented Design subjects this service candidate to additional considerations that shape it into a technical service contract in alignment with other service contracts being produced for the same service inventory. Different approaches are used during this stage for the design of REST services, as these types of services share a common universal contract. For example, the design of media types, the determination of allowable HTTP method usage, and the definition of resource identifiers all require attention during this stage.

As a precursor to the Service Logic Design stage, Service-Oriented Design is comprised of a process that steps Service Architects through a series of considerations to ensure that the service contract being produced fulfills business requirements while representing a normalized functional context that further adheres to service-orientation principles.

Part of this process further includes the authoring of the SLA, which may be especially of significance for cloud-based services being offered to a broader consumer base via the SaaS cloud delivery model.

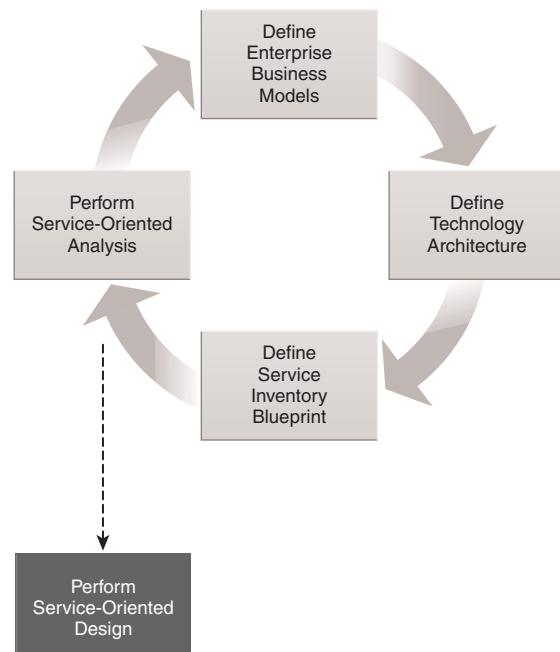


Figure 5.5

When a given service is deemed to have received sufficient analysis effort, it is subjected to the Service-Oriented Design process that produces a physical service contract.

Service Logic Design

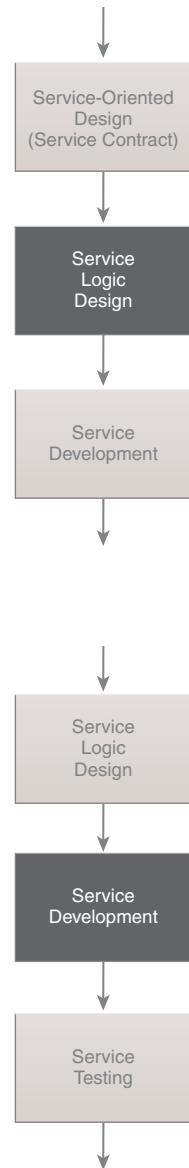
By preceding the design of service logic with the Service-Oriented Design process, the service contract is established and finalized prior to the underlying service architecture and the logic that will be responsible for carrying out the functionality expressed in the service contract. This deliberate sequence of project stages is in support of the Standardized Service Contract (475) principle, which states that service contracts should be standardized in relation to each other within a given service inventory boundary.

The design of service logic and the service architecture is then further influenced by several service-orientation design principles and also whether or not the service will be deployed in a cloud environment (which can impact aspects of the service architecture design, especially when having to accommodate proprietary characteristics imposed by cloud providers).

Service Development

After all design specifications have been completed, the actual programming of the service can begin. Because the service architecture will already have been well-defined as a result of the previous stages and the involvement of custom design standards, service developers will generally have clear direction as to how to build the various parts of the service architecture.

For organizations employing the PaaS cloud delivery model, the service development platform itself may be offered by a ready-made environment hosted by virtual servers and geared toward the development and maintenance of cloud-based services and solutions.

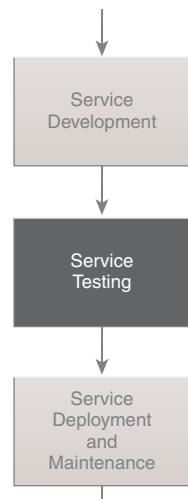


Service Testing

Services need to undergo the same types of testing and quality assurance cycles as traditional custom-developed applications. However, in addition, there are new requirements that introduce the need for additional testing methods and effort. For example, to support the realization of the Service Composability (486) principle, newly delivered services need to be tested individually and as part of service compositions. Agnostic services that provide reusable logic especially require rigorous testing to ensure that they are ready for repeated usage (both concurrently as part of the same service compositions and by different service compositions).

Below are examples of common Service Testing considerations:

- What types of service consumers could potentially access a service?
- Will the service need to be deployed in a cloud environment?
- What types of exception conditions and security threats could a service be potentially subjected to?
- Are there any security considerations specific to public clouds that need to be taken into account?
- How well do service contract documents communicate the functional scope and capabilities of a service?
- Are there SLA guarantees that need to be tested and verified?
- How easily can the service be composed and recomposed?
- Can the service be moved between on-premise and cloud environments?
- How easily can the service be discovered?
- Is compliance with any industry standards or profiles (such as WS-I profiles) required?
- If cloud-deployed, are there proprietary characteristics being imposed by the cloud provider that are not compatible with on-premise service characteristics?
- How effective are the validation rules within the service contract and within the service logic?



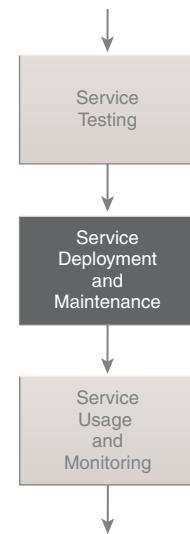
- Have all possible service activities and service compositions been mapped out?
- For service compositions that span on-premise and cloud environments, is the performance and behavior consistent and reliable?

Because services are positioned as IT assets with runtime usage requirements comparable to commercial software products, similar quality assurance processes are generally required.

Service Deployment and Maintenance

Service deployment represents the actual implementation of a service into the production environment. This stage can involve numerous inter-dependent parts of the underlying service architecture and supporting infrastructure, such as:

- distributed components
- service contract documents
- middleware (such as ESB and orchestration platforms)
- cloud service implementation considerations
- cloud-based IT resources encompassed by an on-premise or cloud-based service
- custom service agents and intermediaries
- system service agents and processors
- cloud-based service agents, such as automated scaling listeners and pay-for-use monitors
- on-demand and dynamic scaling and billing configurations
- proprietary runtime platform extensions
- administration and monitoring products



Service maintenance refers to upgrades or changes that need to be made to the deployment environment, either as part of the initial implementation or subsequently. It does not pertain to changes that need to be made to the service contract or the service logic, nor does it relate to any changes that need to be made as part of the environment that would constitute a new version of the service.

Service Usage and Monitoring

A service that has been deployed and is actively in use as part of one or more service compositions (or has been made available for usage by service consumers in general) is considered to be in the Service Usage and Monitoring stage. The on-going monitoring of the active service generates metrics that are necessary to measure service usage for evolutionary maintenance (such as scalability, reliability, etc.), as well as for business assessment reasons (such as when calculating cost of ownership and ROI).

Special considerations regarding this stage apply to cloud-based services. For example:

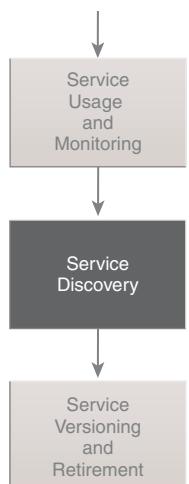
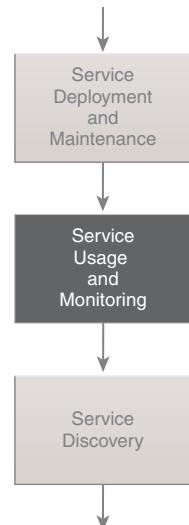
- The cloud service may be hosted by virtualized IT resources that are further hosted by physical IT resources shared by multiple cloud consumer organizations.
- The cloud service usage may be monitored not only for performance, but also for billing purposes when its implementation is based on a per-usage fee license.
- The elasticity of the cloud service may be configured to allow for limited or unlimited scalability, thereby increasing the range of behavior (and changing its usage thresholds) when compared to an on-premise implementation.

This phase is often not documented separately, as it is not directly related to service delivery or projects responsible for delivering or altering services. It is noted in this book because while active and in use, a service can be subject to various governance considerations.

Service Discovery

In order to ensure the consistent reuse of agnostic services and service capabilities, project teams carry out a separate and explicitly defined Service Discovery process. The primary goal of this process is to identify one or more existing agnostic services (such as utility or entity services) within a given service inventory that can fulfill generic requirements for whatever business process the project team is tasked with automating.

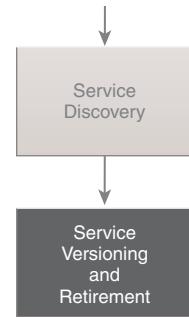
The primary mechanism involved in performing Service Discovery is a service registry that contains relevant metadata about available



and upcoming services, as well as pointers to the corresponding service contract documents, (which can include SLAs). The communications quality of the metadata and service contract documents play a significant role in how successful this process can be carried out. This is why one of the eight service-orientation principles (the Service Discoverability (484) principle) is dedicated solely to ensuring that information published about services is highly interpretable and discoverable.

Service Versioning and Retirement

After a service has been implemented and used in production environments, the need may arise to make changes to the existing service logic or to increase the functional scope of the service. In cases like this, a new version of the service logic and/or the service contract will likely need to be introduced. To ensure that the versioning of a service can be carried out with minimal impact and disruption to service consumers that have already formed dependencies on the service, a formal service versioning process needs to be in place.



There are different versioning strategies, each of which introduces its own set of rules and priorities when it comes to managing the backwards and forwards compatibilities of services. Appendix F provides fundamental coverage of the three most common service versioning approaches (Loose, Flexible, Strict), along with examples for Web services and REST services.

The service versioning stage is associated with SOA governance because it can be a recurring and on-going part of the overall service lifecycle. As explained in Chapter 11, governance approaches that dictate how service versioning is to be carried out will have a significant influence on how a given service will evolve over time. Because this stage also encompasses the termination or retirement of a service, these influences can further factor into the service's overall lifespan.

SUMMARY OF KEY POINTS

- SOA delivery projects contain stages found in traditional IT projects, but also introduce new processes, such as Service Inventory Analysis and Service Discovery.
- One of the main considerations with SOA delivery projects is how much time is spent on the up-front analysis and modeling of services prior to their physical design and development.

5.2 Organizational Roles

To realize the distinct requirements that come with conceptualizing, designing, building, deploying, and evolving services and service-oriented solutions, a correspondingly distinct set of project roles and responsibilities exist.

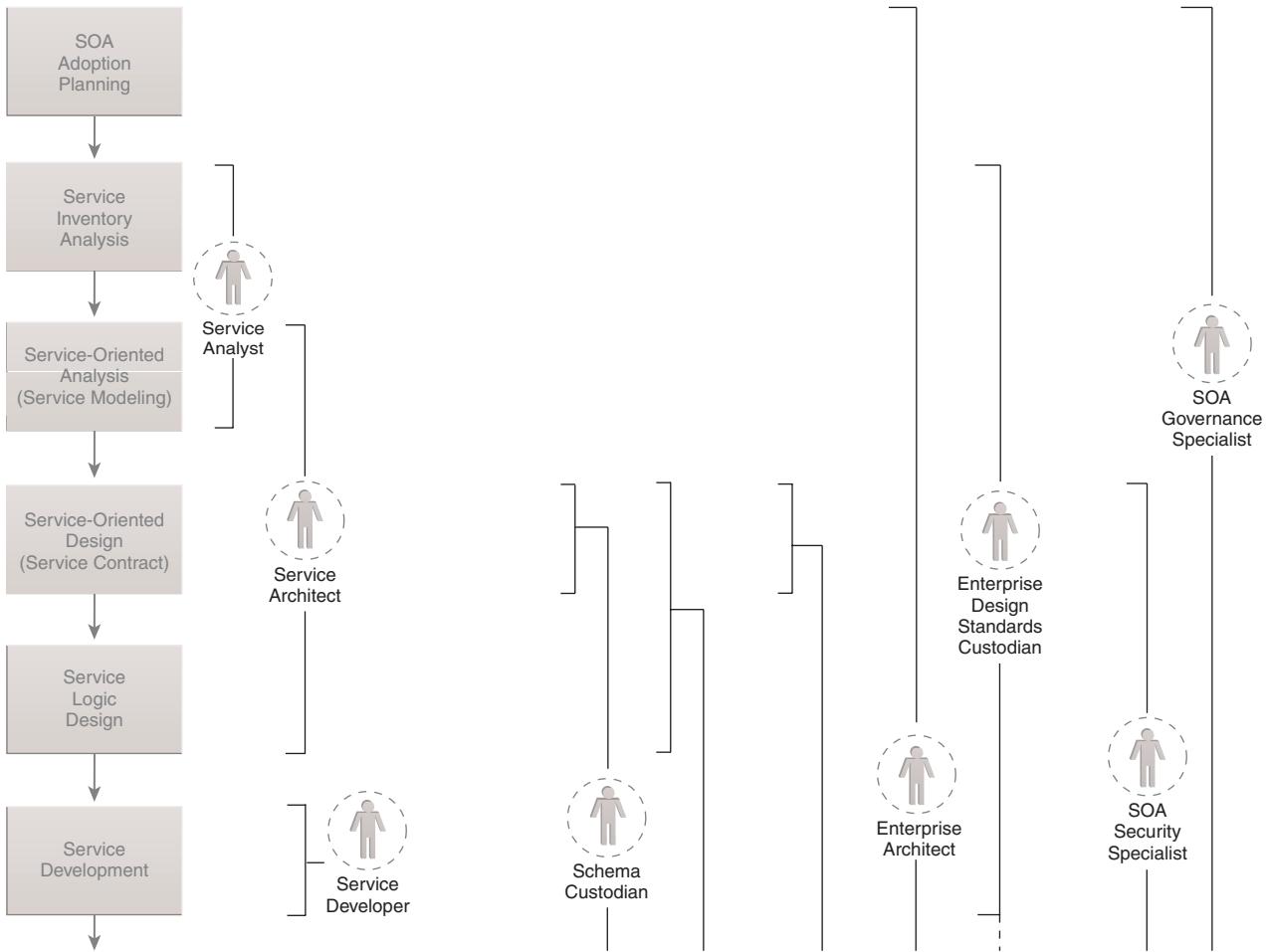
Provided in this section are descriptions for the following set of roles:

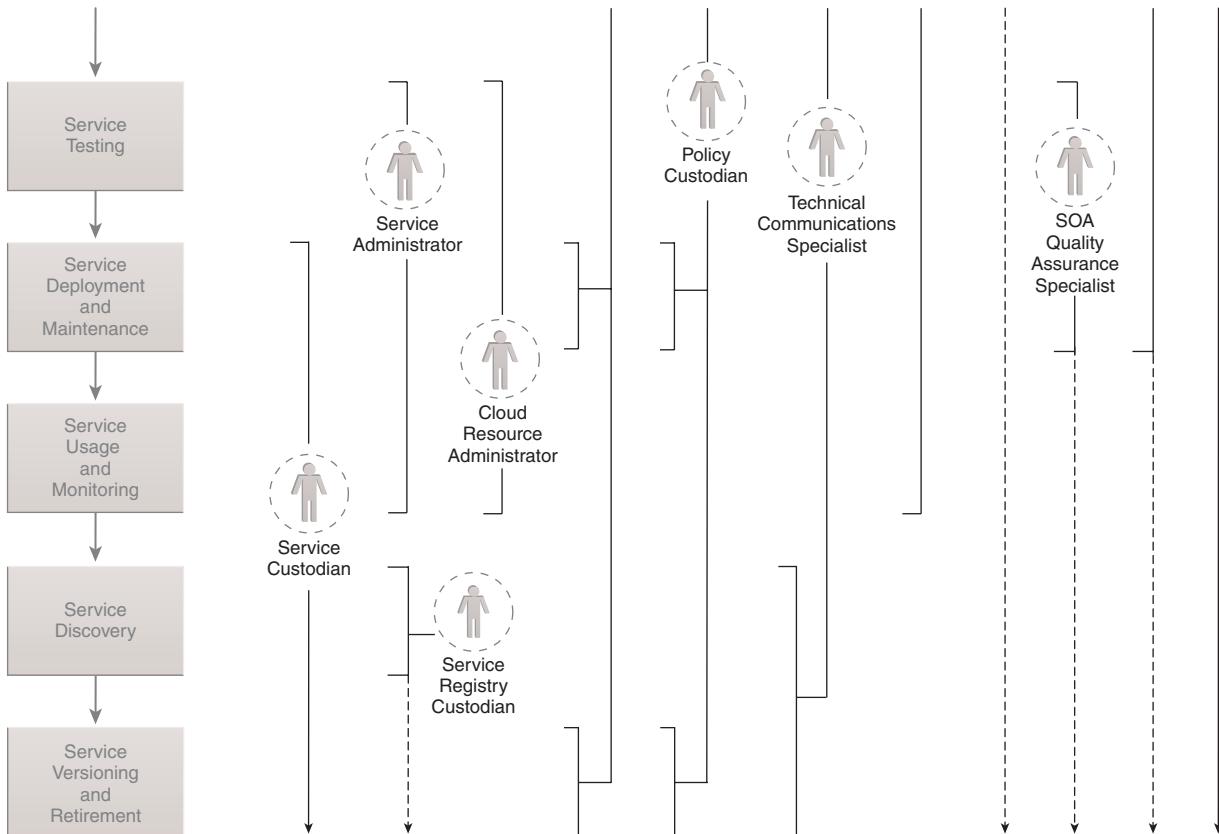
- Service Analyst
- Service Architect
- Service Developer
- Service Custodian
- Cloud Service Owner
- Service Administrator
- Cloud Resource Administrator
- Schema Custodian
- Policy Custodian
- Service Registry Custodian
- Technical Communications Specialist
- Enterprise Architect
- Enterprise Design Standards Custodian (and Auditor)
- SOA Quality Assurance Specialist
- SOA Security Specialist
- SOA Governance Specialist

Note that this list represents *common* roles associated with SOA projects, as well as SOA-related roles within the overall IT enterprise. Variations of these project roles can exist, and additional roles can further be defined. It is also worth noting that a given role can be fulfilled by one or more individuals and that a single individual can be assigned one or more roles.

IMPORTANT

The organizational role/project stage associations displayed in Figure 5.6 are focused on overall project participation and therefore are not specific to governance activities. Governance-specific participation and involvement for each organizational role is described separately throughout Chapters 7 to 12 in this book.



**Figure 5.6**

Shown here are common associations of organizational roles with different SOA project stages.

Service Analyst

The Service Analyst specializes in the areas of Service-Oriented Analysis and service modeling in order to provide expertise in the definition of service candidates, service capability candidates, and service composition candidates.

Service analysis (or SOA analysis) is a dedicated profession. However, the Service Analyst role can also be assumed by architects and Business Analysts that participate in a project's Service-Oriented Analysis phase. Alternatively, it can form the basis of a team leader role within this process; essentially a specialist in Service-Oriented Analysis that coordinates and leads Service Architects and Business Analysts throughout all process steps. The latter variation can be very effective in larger enterprise environments where every iteration through a business process can require the participation of different business and technology subject matter experts.

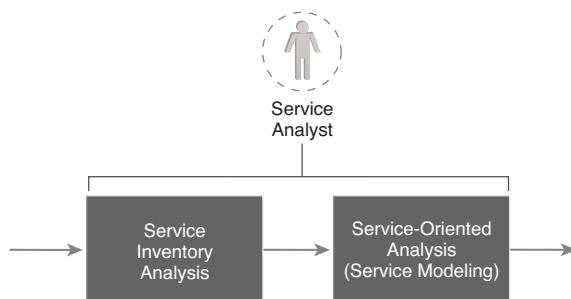
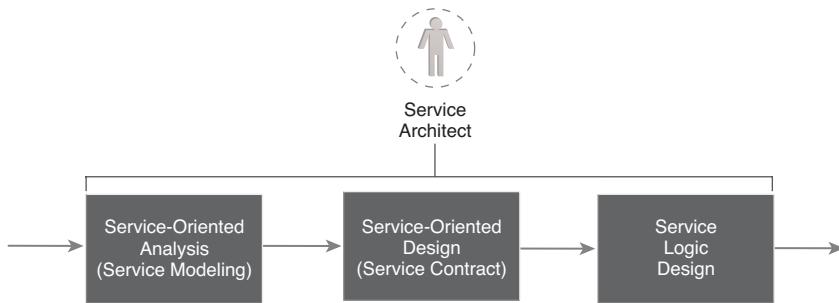


Figure 5.7

Service Analysts are focused primarily on analysis-related stages.

Service Architect

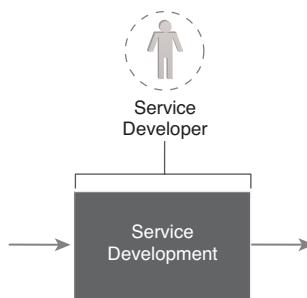
The Service Architect role is concentrated on the physical design of services. Therefore, this role is primarily associated with the Service-Oriented Design and Service Logic Design processes. Service Architects are commonly involved with the Service-Oriented Analysis stage as well. Service Architects therefore can participate in the definition of service candidates and then assume responsibility for authoring design specifications that fulfill business requirements while being compliant to design standards and service-orientation principles.

**Figure 5.8**

Service Architects will usually participate with the definition of service candidates during the Service-Oriented Analysis stage, in addition to their involvement with the actual physical design of service contracts and logic.

Service Developer

A Service Developer is a programmer proficient in the development tools, programming languages, industry markup languages and standards, and related technologies required to build service contracts, logic, and other parts of the service architecture. Service Developers are further versed in the custom design and development standards established by the Enterprise Design Standards Custodian (and others) and are accustomed to developing services in compliance with those standards.

**Figure 5.9**

The Service Developer is primarily involved during the Service Development stage.

Service Custodian

A Service Custodian owns the management and governance responsibilities of one or more specific services. These duties do not just revolve around the extension and expansion and maintenance of service logic, but also include having to protect the integrity of the service context and its associated functional boundary.

Service Custodians are important to the evolution of agnostic services. Their involvement ensures that no one project team inadvertently skews the design of an agnostic service in favor of specific or single-purpose requirements. They are furthermore responsible for hiding non-essential information about service designs from the outside world (as per the access control levels established by the Service Abstraction (478) principle). As a result, Service Custodians often require a good amount of authority.

Note also that depending on how service details are documented, a Service Custodian may author, own, and maintain a service's corresponding service profile document (as explained in the upcoming *Service Profiles* section).

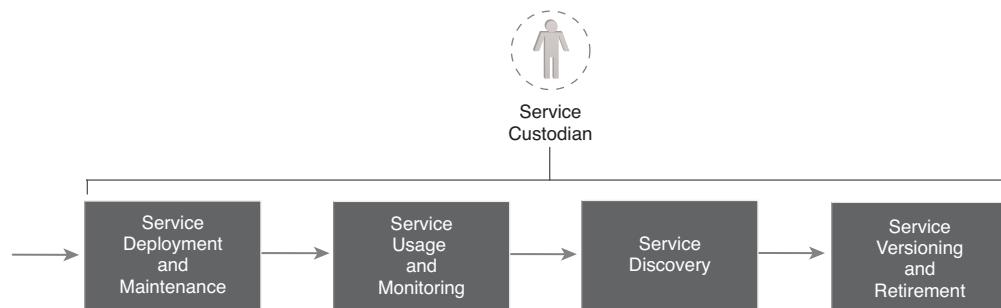
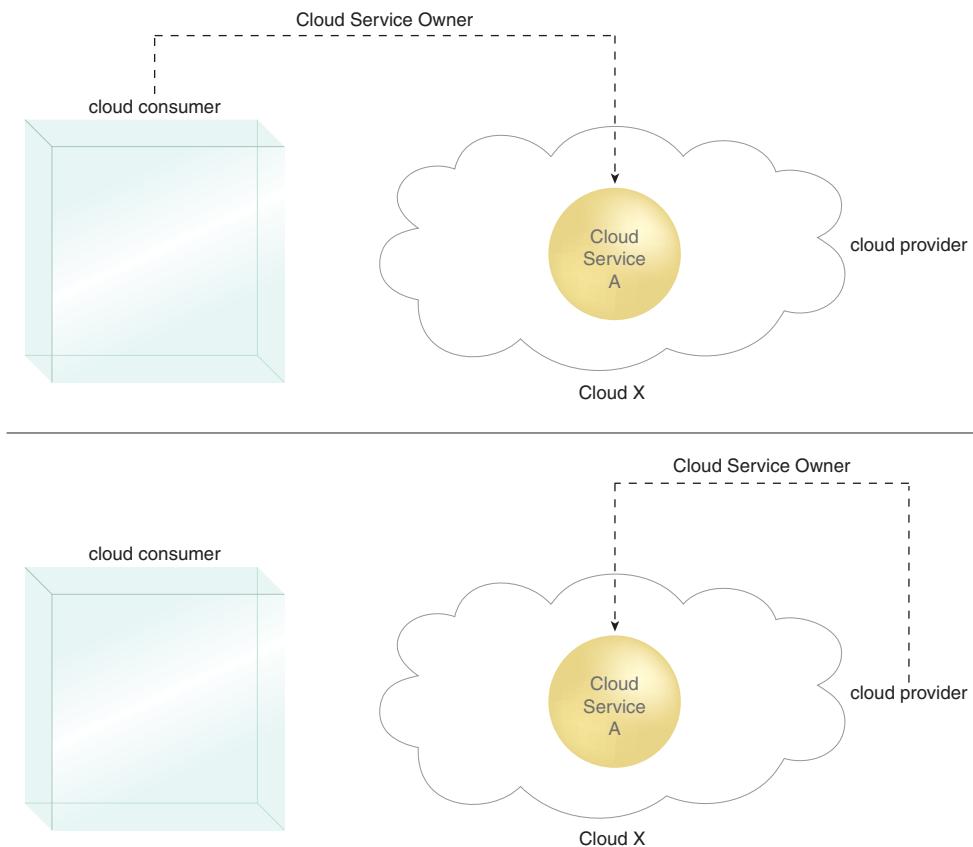


Figure 5.10

Even though a Service Custodian can take ownership of a service as early as when its context is first defined (and verified) during the Service-Oriented Analysis stage, they are typically assigned custodianship upon delivery of the implemented service by the development team.

Cloud Service Owner

A Cloud Service Owner is the person or organization that legally owns a service deployed in a cloud. The Cloud Service Owner can be either the cloud consumer or the cloud provider of the cloud within which the cloud service resides. This distinction is especially relevant when cloud services exist within public clouds owned by third-party cloud provider organizations.

**Figure 5.11**

If Cloud X hosts Cloud Service A then either the organization acting as the cloud consumer of Cloud X can be the Cloud Service Owner (top) of Cloud Service A, or the cloud provider of Cloud X can be the Cloud Service Owner of Cloud Service A (bottom).

NOTE

Unlike other roles described in this chapter, the Cloud Service Owner is not directly mapped to SOA governance precepts and processes in Chapters 7 to 12. This is primarily because this role is used to identify the legal owner of the cloud service, and therefore does not necessarily represent a role actively involved with SOA project delivery stages. For project participation, the Cloud Service Owner will typically engage any number of individuals or groups that fulfill the other described roles, such as the Service Custodian or the Cloud Resource Administrator.

Service Administrator

A Service Administrator is responsible for administering a service implementation. This role is concerned with ensuring that the service and any resources it may share or depend on are properly configured and maintained so that the service's performance is consistent and in line with any guarantees published in its SLA.

The Service Administrator role is typically associated with on-premise service implementations. For services deployed in cloud environments, the Cloud Resource Administrator role is assumed instead.

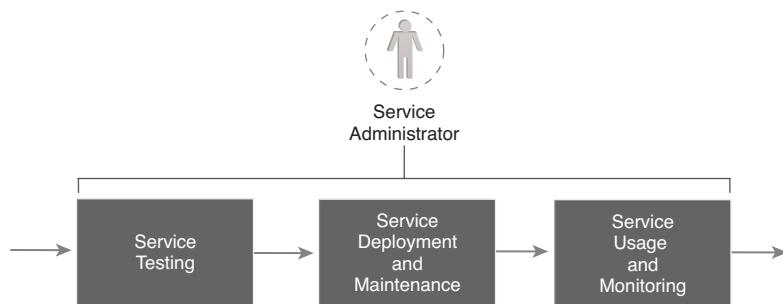


Figure 5.12

The Service Administrator generally gets involved when the service is first deployed in testing and then production environments, and subsequently maintains the service during its runtime existence.

Cloud Resource Administrator

A Cloud Resource Administrator is responsible for administering a cloud service and other types of cloud-based IT resources. This role is proficient with cloud computing technologies and mechanisms and will typically begin its involvement with the initial deployment of a service and then assume the responsibility of maintaining a cloud service implementation in relation to its surrounding cloud-based infrastructure and resources.

A primary concern of the Cloud Resource Administrator is the tuning of on-demand scalability and pay-per-usage mechanisms offered by the cloud environment. In third-party cloud platforms, there may be a variety of options for dynamic scaling and access to shared and virtualized IT resources. Some of these options may have billing implications, while others may introduce performance or behavioral factors.

Besides being dedicated to cloud-based administration requirements, this role is further distinguished from the Service Administrator role in relation to the role's affiliation. A Cloud Resource Administrator can be (or belong to) the organization acting as the consumer or client of a public cloud, or the organization that owns and provides the public cloud.

For example:

- A cloud consumer organization may deploy a service on a public cloud and then assign its own Cloud Resource Administrator to administer that service.
- A cloud provider organization may make a cloud service publically available on a pay-per-usage basis and will therefore have its own Cloud Resource Administrator administer that service. This type of cloud service (which would be categorized as a Software-as-a-Service offering) could then be used as part of a service composition developed by the cloud consumer organization.

Also worth noting is that Cloud Resource Administrators can be (or belong to) a third-party organization contracted to administer the cloud-based service or IT resource. For example, a Cloud Service Owner could outsource administration responsibilities to a Cloud Resource Administrator to administer its cloud service.

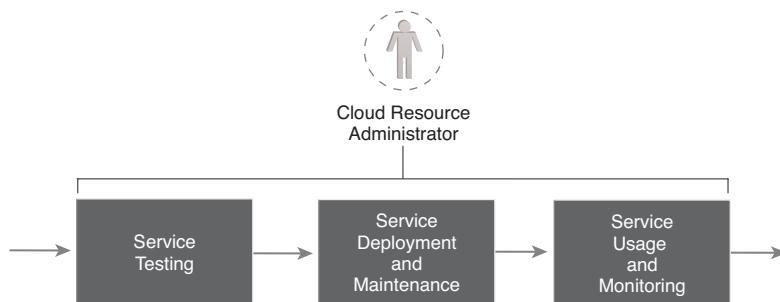


Figure 5.13

As with the Service Administrator, this role is generally involved from Service Testing through to Service Usage and Monitoring stages.

The reason this role is not named Cloud Service Administrator is because it may be responsible for administering cloud-based IT resources that don't exist as cloud services. For example, if the Cloud Resource Administrator belongs to (or is contracted by) the cloud provider organization, IT resources not made remotely accessible may be

administered by this role (and these types of IT resources are not classified as cloud services). Figure 5.14 illustrates this in relation to the Cloud X and Service A scenario from Figure 5.11.

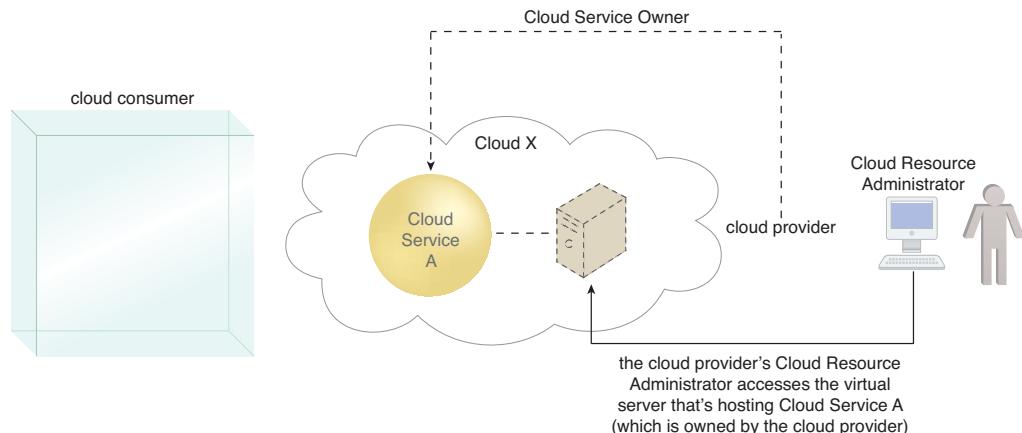


Figure 5.14

While Cloud Service A may be accessible to external cloud consumers, the cloud provider that acts as the Cloud Service Owner can have a Cloud Resource Administrator configure the underlying virtual server that hosts Cloud Service A.

NOTE

In the previous example, the virtual server is considered an IT resource. See Chapter 3 for a definition of the term “IT resource” and for further descriptions of fundamental cloud computing terms.

Schema Custodian

Schema Custodians are primarily responsible for ensuring that service contract schemas (and schemas used elsewhere as the basis of messaging data models), as well as custom media types for uniform contracts, are properly positioned as standardized and centralized parts of service inventories. Schema Custodians may even own design standards pertaining to service contracts and relevant data modeling.

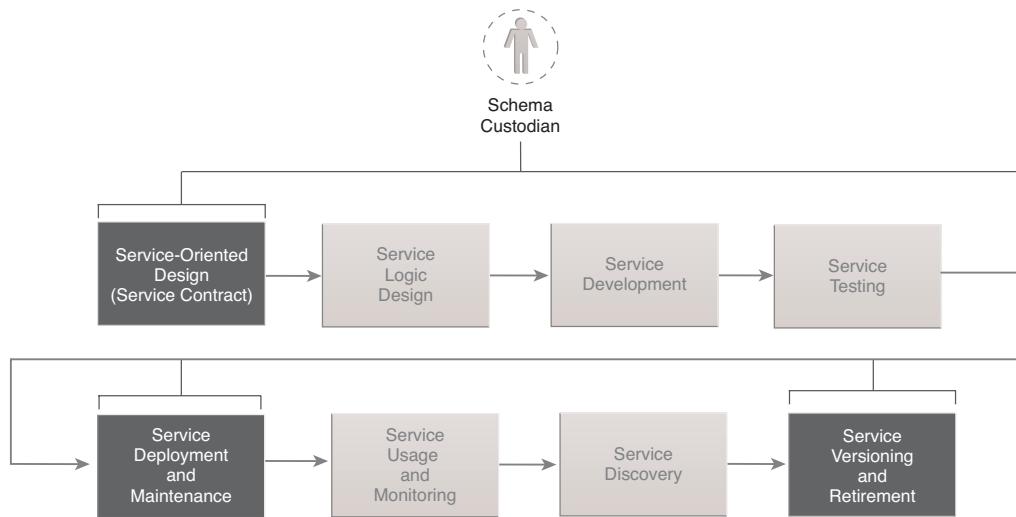


Figure 5.15

The Schema Custodian gets involved whenever the service contract's schema(s) are affected. It is part of its definition, implementation, and any subsequent versioning requirements. The Schema Custodian role was originally established in the book *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*.

“XML Schema” vs. “XML schema”

The word “schema” is capitalized when referring to the XML Schema language or specification, and it is lower case when discussing schema documents.

For example, the following statement makes reference to the XML Schema language:

“One feature provided by XML Schema is the ability to...”

And this sentence explains the use of XML schema documents:

“When defining an XML schema it is important to...”

Note also that we often refer to XML schemas as just “schemas.”

Policy Custodian

The Policy Custodian role is assigned the responsibility of defining and maintaining machine-language (technical) and natural-language (human-readable) policies used by service contracts and policies assigned to services in general. Although this role can be assumed by the same person acting as a Schema Custodian, it is not uncommon for different individuals (or even different groups) to be responsible for defining and maintaining policies.

Policy Custodians are often required to ensure that operational service policies remain in alignment with parent business policies. This can further lead to unique versioning requirements as business policies can be subject to various changes that end up affecting service contracts.

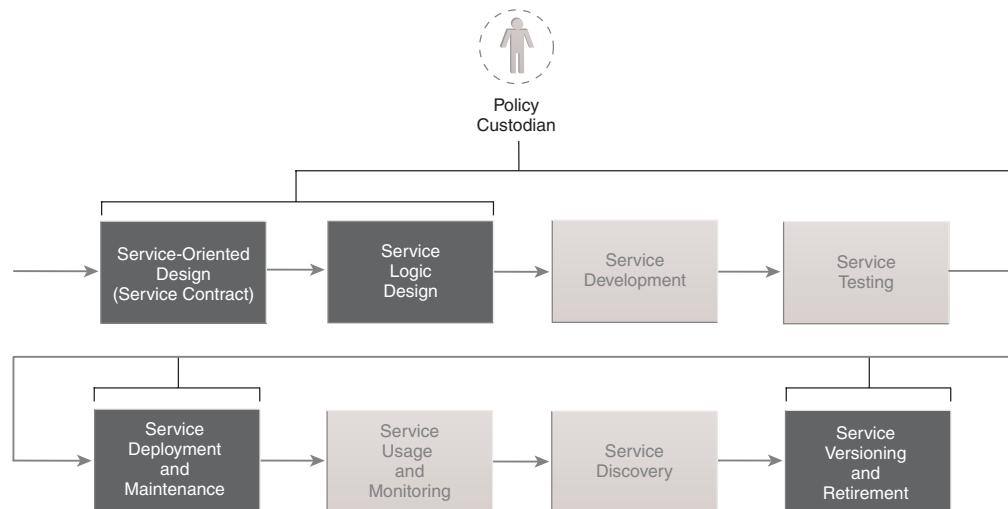


Figure 5.16

As with the Schema Custodian, this role is closely tied to the service contract. In this case, it is involved when service policies related to the service contract are affected.

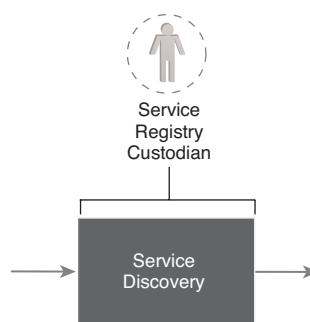
Service Registry Custodian

The Service Registry Custodian is tasked with the overall administration of one or more service registries. This goes beyond the installation and maintenance of the registry product; it encompasses the constant responsibility of ensuring a high quality of registry record content, which ties directly into how discoverability-related meta information is defined and recorded for individual services.

A Service Registry Custodian needs to ensure that once a service registry is introduced into an IT enterprise, it is consistently maintained to avoid becoming stale or inaccurate. Although Service Registry Custodians will typically not author discoverability content themselves, they will often own standards or conventions that dictate the nature of metadata used to populate service registry profile records.

Figure 5.17

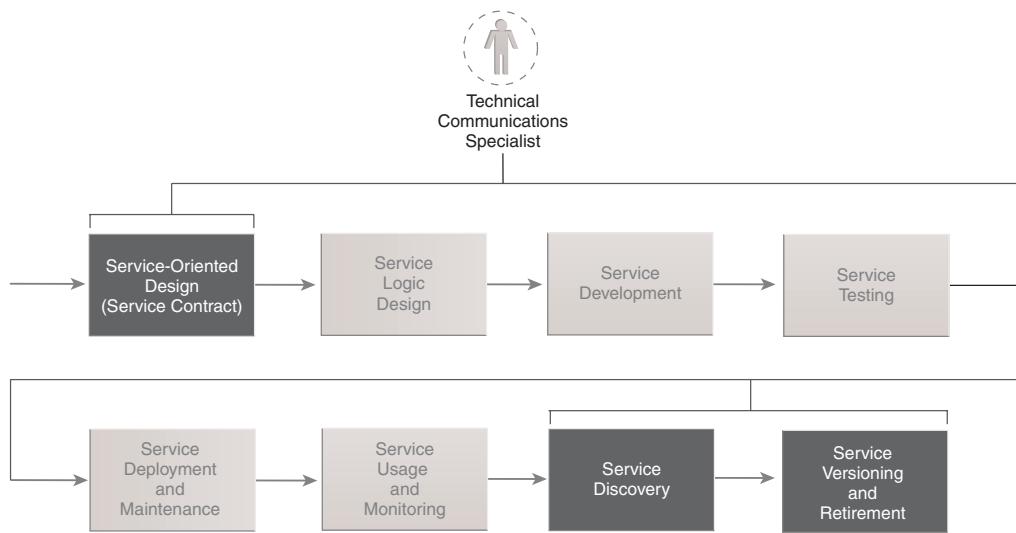
This role is focused almost exclusively in the Service Discovery stage. It is further affected when new versions of a service or service contract impact what is recorded in the service registry (which, in turn, eventually affects the Service Discovery stage again). Note that registry products are discussed in Chapter 14.



Technical Communications Specialist

A Technical Communications Specialist is usually someone with a background in technical writing who is enlisted to refine initial drafts of service profiles and associated service contracts and metadata.

The responsibility of this role is to express discoverability information, using standard vocabularies so that a range of project team members can effectively query and interpret service contracts and associated profiles. This makes information published about a service accessible to a broader range of project team members and reduces risk associated with misinterpretation and inadvertent non-discovery.

**Figure 5.18**

Stages pertaining to the authoring, definition, or expansion of content associated with what is published about a service are relevant to this role.

Enterprise Architect

Although this is not a new role by any means, it represents a position that is greatly emphasized by the cross-application (cross-silo) scope of service inventory delivery projects.

Technology Architects with an enterprise perspective are expected to:

- assist with SOA Adoption Planning and strategy
- author or contribute to enterprise design standards
- become involved in service delivery projects to ensure that agnostic services are properly positioned
- assess service runtime usage and determine required infrastructure
- evaluate security concerns of individual service capabilities
- help define and perhaps even own service inventory blueprints

In larger organizations there may also be the need for Enterprise Domain Architects—a variation of this role that specializes in a particular segment of the overall enterprise. These architects would then be focused on the definition and governance of domain-specific service inventories.

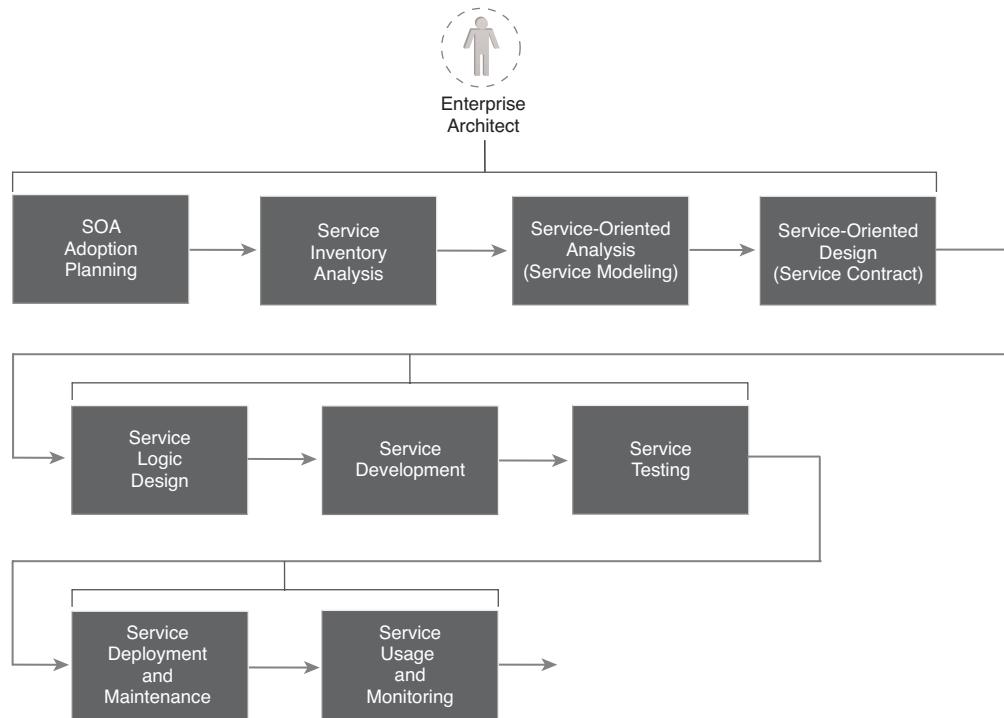


Figure 5.19

Enterprise Architects can be involved with just about every stage.

Enterprise Design Standards Custodian (and Auditor)

As enterprise architecture groups grow in response to the changes incurred by an SOA transition, design standards can be authored by multiple experts, each contributing conventions associated with a particular aspect of service design (security, performance, transactions, uniform contract usage, caching, etc.).

To ensure that design standards are kept in alignment and used wherever appropriate, it may very well be necessary to establish an official custodian for enterprise design specifications. This individual or group is responsible for the evolution of the design standards, as well as their enforcement. Therefore, this role often involves performing audits of proposed service or service-oriented solution designs.

The authority required to carry out auditing responsibilities can sometimes raise concerns within IT environments not accustomed to such formal use of design standards.

Therefore, this role can be more successfully established within the boundaries of a specific enterprise domain, where a given set of standards applies only to a specific domain service inventory, not the enterprise as a whole.

The authority needed to audit and *enforce* the use of standards is a requirement for this role to be carried out successfully. This essential requirement relates back to the *Discipline* sub-section of *The Four Pillars of Service-Orientation* section in Chapter 4.

NOTE

Although the official title for this role is Enterprise Design Standards Custodian (and Auditor), for the sake of brevity, this role is referred to as just Enterprise Design Standards Custodian throughout this book.

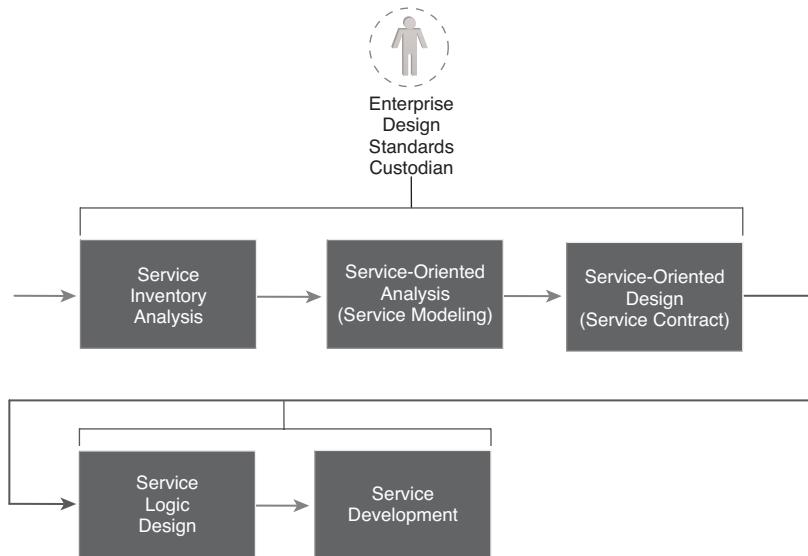


Figure 5.20

Although design standards can impact any part of a service's lifecycle, this role is primarily concerned with its initial delivery.

SOA Quality Assurance Specialist

This role is comparable to the traditional IT quality assurance profession, except that it requires further expertise in ensuring the quality of services—in particular, shared services that can be subjected to repeated reuse and composition.

SOA Quality Assurance Specialists are closely associated with the Service Testing stage, during which services can be subjected to various forms of tests either prior to their initial deployment or as part of the release of subsequent service versions.

NOTE

Several of the governance precepts and processes covered in this book are related to “ensuring the quality” of some aspect of SOA and the delivery of services. Although SOA Quality Assurance Specialists can be involved in these areas (as described in upcoming chapters), these types of quality assurance tasks are primarily associated with the SOA Governance Specialist role.

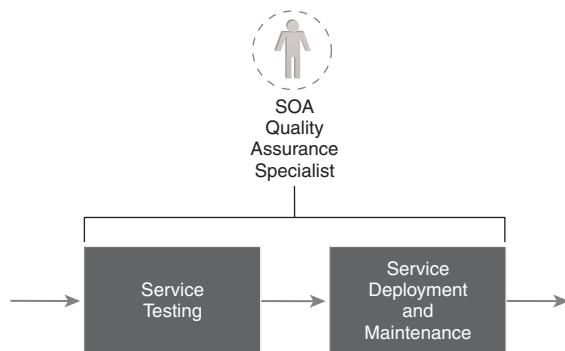


Figure 5.21

The SOA Quality Assurance Specialist usually gets involved during the Service Testing stage and can then further be required to ensure that required levels of quality are actually being met during subsequent stages.

SOA Security Specialist

Each service can have its own individual security requirements and security architecture. Further, with shared agnostic services, the requirements of the service compositions reusing the service to automate different types of business processes also need to be taken into account. In this case, the service may need to join a service composition architecture that introduces a parent security architecture that encompasses the service's individual security architecture.

Either way, the SOA Security Specialist role is dedicated to ensuring that services (individually and as part of service compositions) are properly secured and also that agnostic services are sufficiently flexible so that they can be incorporated into other security architectures when required.

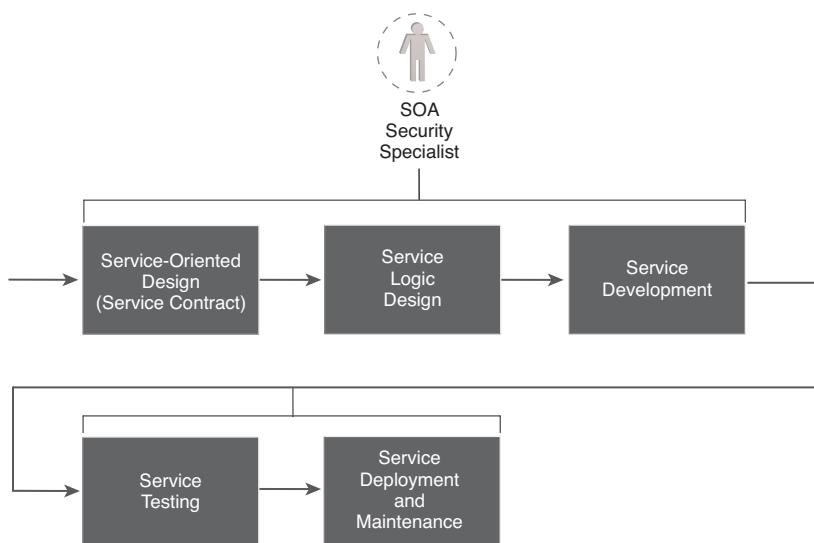


Figure 5.22

Security issues are of primary concern when physical design stages are entered, through to when the service is implemented. SOA Security Specialists will further be involved with subsequent stages, as new security requirements or threats surface.

SOA Governance Specialist

The SOA Governance Specialist is an expert in the definition and execution of governance precepts and processes, as well as the usage of related governance technology. All topics covered in this book pertain to this role.

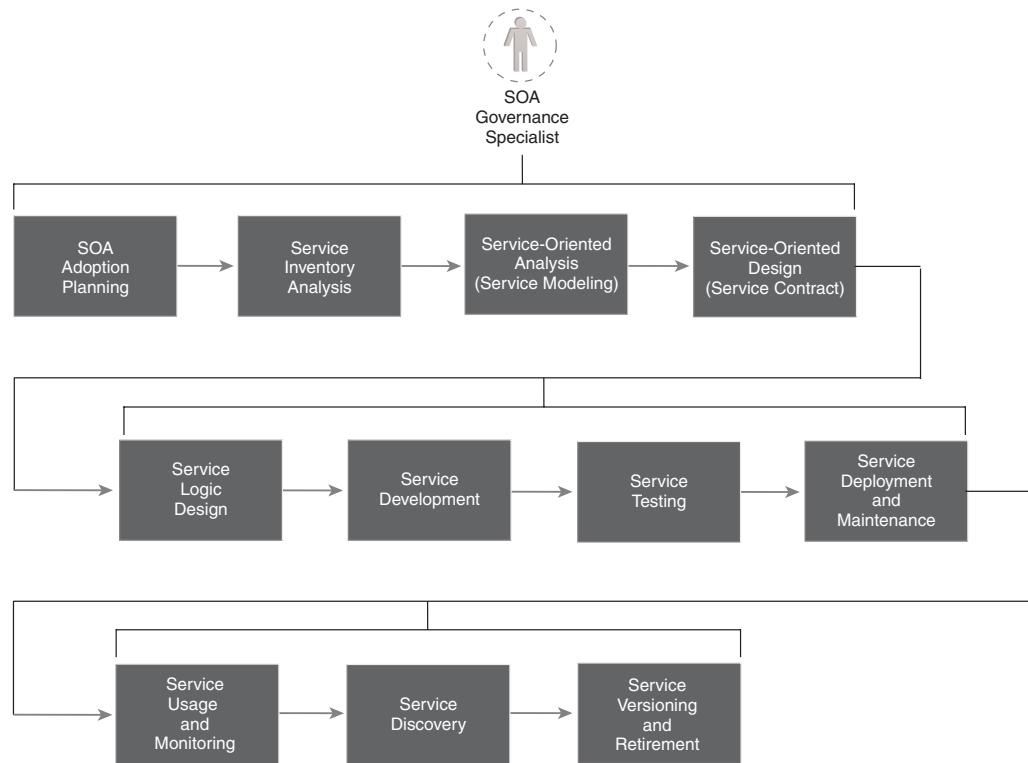


Figure 5.23

This role can be involved with all stages.

NOTE

Individuals assuming the SOA Governance Specialist, Enterprise Architect, and Enterprise Design Standards Custodian roles will generally end up joining the SOA Governance Program Office (explained in Chapter 6).

Other Roles

The following additional roles are also referred to in this book. These brief descriptions are by no means official definitions; they simply introduce the roles in relation to SOA project involvement. IT professionals focused on any of these areas can also assume one or more of the previously explained organizational roles.

NOTE

In the *People (Roles)* sections throughout Chapters 7 to 11, these roles are intentionally not always mapped to SOA governance precepts and processes.

Educator

A fundamental role that must be established and fulfilled on an on-going basis throughout SOA project stages is that of the Educator. Education (one of the four foundational pillars of service-orientation described in Chapter 4) is a critical success factor that impacts the successful fulfillment of *all* the other organizational roles described in this book, in terms of their involvement during SOA project activities, governance-related or otherwise.

Educators act as trainers (delivering instructor-led courses and workshops specific to the skill-set development requirements of a given SOA project) and as mentors (working with SOA project members individually to address knowledge gaps and other mismatches between their backgrounds and assigned responsibilities).

The role of the Educator is not mentioned in subsequent chapters because it is assumed to be ever-present. The Educator is there during the early stages to teach what project teams need to learn in preparation for a given SOA project. The Educator is then further there and available when the project commences to provide guidance, mentorship, and to deliver additional workshops, as required (such as when a new technology or tool is chosen to replace one originally selected during the planning stages).

In some cases, the SOA Governance Specialist may assume the role of a quasi-Educator when providing assistance to other project team members with the application and usage of SOA governance controls. However, to truly assume the role of Educator requires structured course development, dedicated tutorship, superior communication, and, typically, instructor-level accreditation. Therefore, an Educator specializing in SOA governance can exist separately from an SOA Governance Specialist.

Business Analyst

The job of a Business Analyst does not change drastically when defining business-centric services. While different organizations may have different business analysis practices and approaches, the primary objectives of this role as part of SOA projects are the creation of business use cases, business requirements, and the definition of business processes.

In relation to service modeling, Business Analysts are responsible for creating the business processes specifications that act as input for the Service-Oriented Analysis process. They are further expected to participate in the Service-Oriented Analysis process for the definition of business-centric service candidates.

Data Architect

Data Architects provide guidance for the identification of the data necessary for a service to comply with functional and non-functional requirements. They are usually also responsible for defining and maintaining the data models used to establish the structure of messages and the validation logic that can reside within service contracts and the underlying service implementations. As a result, it is fairly common for the Schema Custodian role to be assigned to a Data Architect.

NOTE

For the purpose of this book only, the term “Data Architect” encompasses information architects, data modelers, and data analysts.

Technology Architect

Although Service Architects will tend to already be Technology Architects, a distinction between these two roles is still necessary, especially when services need to access or encapsulate other IT resources. There may be Technology Architects (that are not qualified Service Architects) that need to be consulted or involved during SOA projects.

NOTE

For the purpose of this book only, the term “Technology Architect” encompasses application architects, systems architects, solution architects, and software architects.

Cloud Technology Professional

Any IT professional that is required to build, deploy, or work with cloud-based services and IT resources needs to be proficient with the core technologies, technical mechanisms, and fundamental security concerns that are relevant to contemporary cloud environments. The Cloud Technology Professional has obtained a proven understanding of the building blocks that comprise cloud solutions and enable key characteristics, such as elasticity and virtualization, and has further demonstrated ability to identify and address common security threats specific to cloud environments. This role is assumed by anyone working hands on with cloud-based IT resources, and is therefore typically combined with other roles, such as Service Developer and Service Administrator.

Cloud Architect

Cloud Architects specialize in cloud-based technology architecture, design patterns, mechanisms, and are proficient with contemporary analysis and design processes for authoring detailed architectural specifications of cloud-based solutions and platforms. The Cloud Architect's skill-set is not limited to cloud environments; this role is required to architect solutions and service compositions that span on-premise and cloud platforms and to further design dynamic and complex solutions using remote distribution techniques, such as cloud bursting and cross-cloud balancing. This role is most commonly coupled with the Service Architect role.

Cloud Security Specialist

These are IT professionals with expertise specific to security threats (and mechanisms used to counter those threats) pertaining to cloud-based services and supporting cloud-based IT resources. Cloud Security Specialists are distinct from SOA Security Specialists, in that they are focused solely on security issues related to cloud environments and IT resources, regardless of whether the underlying solutions are service-oriented or based on service-oriented technology architectures.

Cloud Governance Specialist

Public, private, and community clouds each have their own distinct characteristics and platforms, many of which are proprietary and vendor-specific. Amidst the diversity of these environments, there is a set of mechanisms and goals that are common to most initiatives that involve or are based on the use of cloud computing. Cloud Governance

Specialists are proficient in establishing IT governance precepts and processes in support of regulating general and proprietary mechanisms, technologies, solutions, and IT resources that reside and operate within cloud environments in order to ensure that the planned goals are attained.

This role is distinct from the SOA Governance Specialist role in that its focus is on regulating cloud environments and cloud-based IT resources, regardless of whether they are being used by or are associated with service-oriented solutions or service-oriented technology architectures.

IT Manager

IT management's primary focus is to manage IT resources and budgets, ensure proper staffing levels, and set the strategic direction for the department or group for which the IT Manager is responsible. Traditional IT Managers can be impacted by SOA initiatives and the need for new forms of IT Managers can further emerge.

SUMMARY OF KEY POINTS

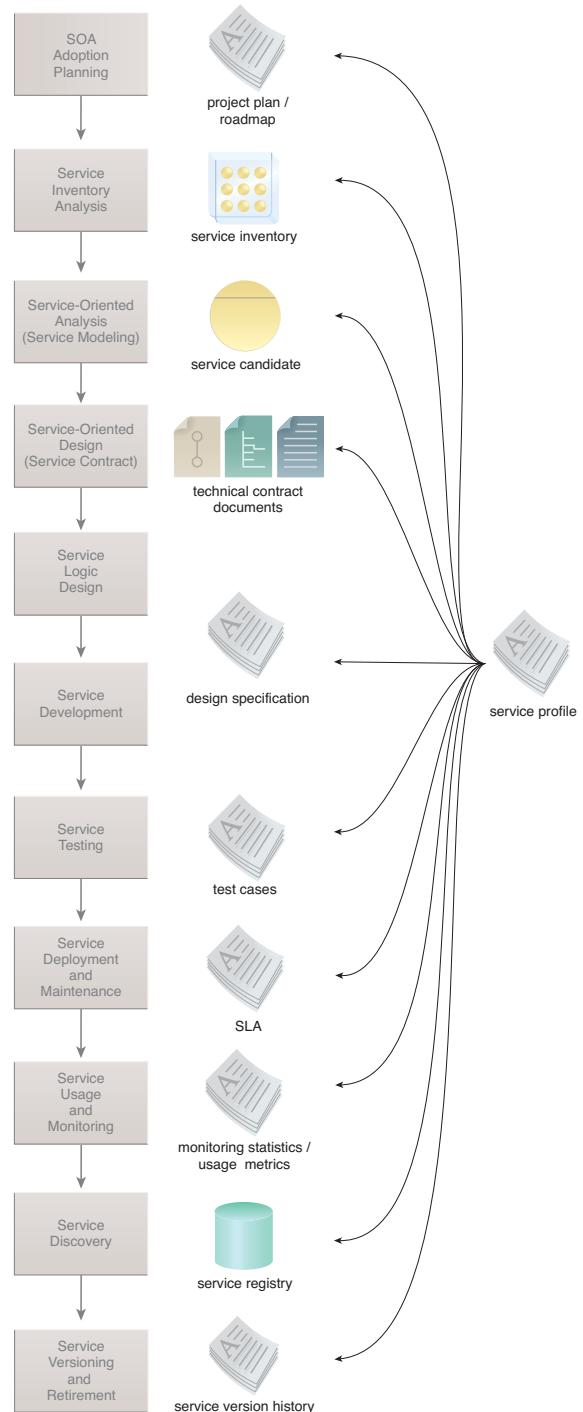
- SOA projects introduce a number of unique project roles, several of which emerged from distinct SOA project and service lifecycle stages.
 - One individual can assume one or more roles and a single role can be assumed by one or more individuals.
 - The SOA Governance Specialist role is associated with all stages.
-

5.3 Service Profiles

When collecting discoverability-related meta information, it is helpful to use a standardized template or form that ensures the same type of data is documented for each service. This can be especially useful during the early analysis stages, when service candidates are just being conceptualized as part of the service modeling process. The document used to record details about a service is the *service profile* (Figure 5.24).

Figure 5.24

A service profile initially acts as a repository of meta information when a service is first conceptualized during early analysis stages, and then later provides valuable details for design and delivery-related documents used during subsequent lifecycle phases.



Service-Level Profile Structure

There is no one official industry format for service profiles. However, once in use, the service profile format must be standardized as part of the SOA initiative. With an understanding of how services typically transition and evolve throughout stages, the following baseline parts and fields are recommended:

- *Service Name*
- *Purpose Description (Short)* – A concise, one sentence description of the service context and purpose.
- *Purpose Description (Detailed)* – A full explanation of the service context and its functional boundary with as many details as necessary.
- *Service Model* – Entity, Utility, Task, Orchestrated Task, or a custom variation.
- *Capabilities* – The profile should document capabilities that exist and are in development, as well as those that are only planned and tentatively defined. Color coding is often useful to make these distinctions as is the use of the capability “status” field (described shortly).
- *Keywords* – This field can contain one or more keywords ideally taken from an official service inventory-level taxonomy or vocabulary. Service profile keywords should correspond to the keywords used by a service registry.
- *Version* – The version number of the service currently being documented is noted here. Depending on the version control system in use, version numbers may only be applicable to service capabilities.
- *Status* – The development status of the service (or service version) is expressed in this field using standard terms identifying a project lifecycle stage, such as “analysis,” “contract design,” “development,” or “production.” If the service is not in production, it can be helpful to include an estimated delivery date.
- *Custodian* – Details on how to reach the official Service Custodian or owner, as well as others that contributed to this documentation.

Capability Profile Structure

Because a service acts as a container for a collection of capabilities, additional “sub-profiles” need to be established to represent each individual capability separately, as follows:

- *Capability Name*
- *Purpose Description* – A concise explanation of the capability’s overall purpose and functional context (similar to the short service description).
- *Logic Description* – A step-by-step description of the logic carried out by the capability. This can be supplemented with algorithms, workflow diagrams, or even entire business process definitions, depending on what stage the capability definition is at.
- *Input/Output* – These two fields provide definitions of a capability’s allowable input and/or output value(s) and associated constraints. It can be helpful to describe these in plain English during the service modeling phase. The details established here can make reference to existing schema types.
- *Composition Role* – The execution of capability logic can place a service into various temporary runtime roles, depending on its position within service composition configurations. This field can be filled out with a description of the capability’s role or it can simply contain a term used to identify predefined runtime roles.
- *Composition Member Capabilities* – A list of services (and specifically their capabilities) composed by the capability logic. This provides a convenient cross-reference to other service capabilities the current capability has formed dependencies on. Ideally, identified composition member capabilities are mapped to the portions of the business process logic (documented in the *Logic Description* field) so that delegated logic is clearly indicated.
- *Keywords* – Often the same keywords that apply to the service can be carried over to the capability. But it is not uncommon for additional keywords to be added to individual capabilities so as to better classify their purpose. Keywords for services and capabilities should originate from the same parent vocabulary.
- *Version* – Depending on the versioning system in place, capabilities themselves may be versioned with a number or new capability versions may be added with the version number appended to the capability name.

- *Status* – The same lifecycle identifiers used for services can be applied to the status of individual capabilities. However, this field can also be used to earmark capabilities that were identified during the modeling stage, but for which no specific delivery date exists.
- *Custodian* – More often than not, the custodian of the service will be the custodian (or one of the custodians) of the related capabilities. However, when multiple business and technology experts collaborate on a given service, some are only there to assist with the definition of one service capability (or a subset of service capabilities). In this case separate custodians may need to be associated with individual capabilities.

Additional Considerations

Customizing Service Profiles

What we've established so far is fundamental profile documentation. Organizations are encouraged to customize and extend this to whatever extent required.

Service Profiles and Service Registries

Much of the information assembled into service profiles will form the basis for service registry records. Depending on whether a service registry exists within an organization at the time the profile is being defined, it is advisable to become familiar with the registry product's record format. This will allow you to better align the service profile template with how the profile information may need to be represented within the service registry.

Service Profiles and Service Catalogs

The structure of a service profile is ideally standardized so that different project teams consistently document the services they deliver. As more service profiles are created, they can be assembled into a *service catalog*. A service catalog is essentially a documentation of the services within a service inventory (much the same way a product catalog may describe the inventory of items a company may have in its warehouse). If an organization is creating multiple domain service inventories, each with its own design standards and governance processes, then service profile structures may vary. Therefore, a separate service catalog is generally created for each service inventory.

Service Profiles and Service Architecture

It's important to keep a clear separation between what is documented in a service profile document and the content of a service architecture specification. The latter form of document is authored by qualified Service Architects and contains the details of the service's design and implementation. As with the service profile, a service architecture specification can be a living document that evolves during the service lifecycle. However, few actual technical implementation details make their way into the corresponding service profile. It is up to the discretion of the Service Custodian (and the Service Architects and others involved with governing the service) to determine what extent of overlap is necessary. For example, it may be necessary to indicate that the service is located in a particular cloud environment and bound to pay-for-use leasing terms. This information is typically relevant to both service profile and service architecture documents. On the other hand, the fact that one of the service capabilities contains logic that accesses a legacy database may only belong in the service architecture specification.

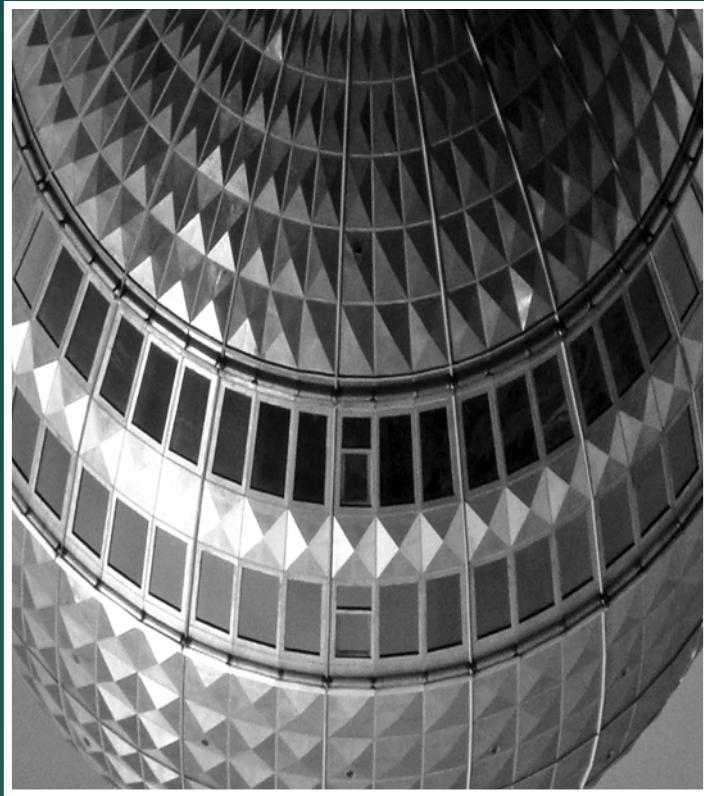
NOTE

Service profiles were first introduced in the book *SOA Principles of Service Design*. See Chapter 15 of that book for a detailed example of a service profile.

SUMMARY OF KEY POINTS

- As services move from concept to candidate to physical design to production usage, it is important to consistently document them using standardized service profiles.
 - The use of service profiles is most effective when combined with a standardized vocabulary or taxonomy.
 - Multiple service profile documents can be compiled to create a service inventory-specific service catalog.
-

Chapter 6



Understanding SOA Governance

6.1 Governance 101

6.2 The SOA Governance Program Office (SGPO)

6.3 SGPO Jurisdiction Models

6.4 The SOA Governance Program

The expectation when adopting service-orientation is the realization of a number of specific strategic business benefits, as explained in Chapter 3. To accomplish this requires not only sound technology, mature practices, and sufficient stakeholder support, but also a firm grasp of the strategic target state being realized by the adoption and a firm system of ensuring its attainment and sustainment. Such a system cannot be purchased with technology products labeled as governance tools; it is a system that requires careful definition specific to overarching goals and requirements.

Structured governance is required to carry out and see through the commitments made when embarking on an SOA roadmap. It helps organizations succeed with SOA adoption efforts by mitigating risks through predefined constraints, rules, and the allocation of necessary authority. This chapter provides an introduction to general governance concepts and terms, as well as fundamental topics regarding governance systems for SOA projects.

6.1 Governance 101

Governance is the act of governing or administrating something. By far the most common form of governance is that of an organization. A system of governance is therefore generally a type of organizational system. For example, a society uses an organizational system to govern a public community. A company uses an organizational system to govern its own internal community.

A system for organizational governance exists as a meta-decision system. In other words, it is not just a means by which the organization makes decisions, it is the means by which the organization makes decisions *about* decision-making.

Within this context, a governance system:

- places constraints on decisions
- determines who has responsibility and authority to make decisions
- establishes constraints and parameters that control, guide, or influence decisions
- prescribes consequences for non-compliance

At the highest level in society, governance is established by a constitution. Within a company, it may be declared in the form of a business charter. Founding documents such as these establish a parent level of authority and constraints from which all other decision-making authorities and structures are derived. At deeper levels within the organization, a governance system can further influence the definition of policies, standards, and processes that guide and control day-to-day decision-making activities.

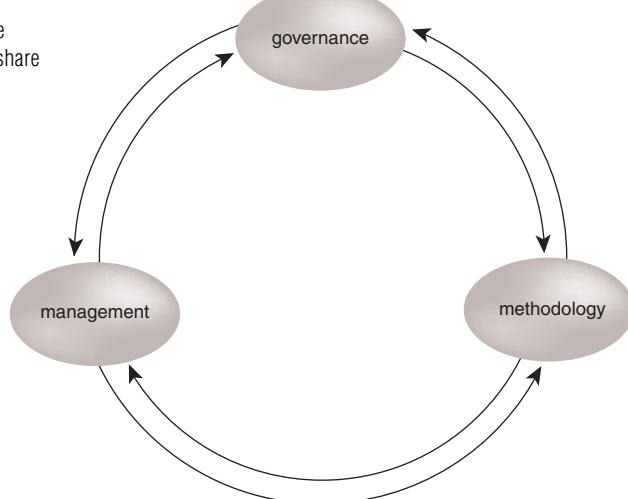
A good system of governance helps the members of an organization carry out responsibilities in a manner supportive of the organization's business goals and vision. It mitigates conflict by clearly defining responsibilities and assignments of authority, and further reduces ambiguity by articulating constraints and parameters in practical forms (such as rules and decision guidelines). It also helps balance tactical and strategic goals by expressing the intents and purposes of its rules.

The Scope of Governance

Within IT, a governance system is responsible for providing organization, direction, and guidance for the creation and evolution of IT assets and resources. To fully understand the scope of a governance system within a given IT department, we need to determine how a governance system relates to and is distinguished from methodology and management (Figure 6.1).

Figure 6.1

Governance, management, and methodology are distinct areas within an IT department that also share distinct relationships.



Governance and Methodology

Methodology represents a system of methods. Within IT, the form of methodology we are generally concerned with is that used to create software programs and business automation solutions. In this context, the methodology determines a system of methods used to conceptualize, design, program, test, and deploy a software program. These methods are generally formalized as a series of step-by-step processes that correspond to project delivery lifecycle stages.

NOTE

The Mainstream SOA Methodology (MSOAM) has established itself as a common, generic methodology for SOA project delivery. This methodology is explained in parts throughout the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*, and is further summarized at www.soamethodology.com. Appendix G provides a supplementary paper that maps MSOAM to the Rational Unified Process (RUP).

Different software delivery methodologies exist. What commonly distinguishes one from the other is how they prioritize tactical and strategic requirements in relation to overarching business goals. These priorities will usually result in different processes (project lifecycle stages) being combined or organized in different ways. In some cases, one methodology may introduce a new process that does not exist in other methodologies—or it may exclude a process that commonly exists in other methodologies. Frequently, however, it comes down to how much time and effort a given process or project lifecycle stage receives, as determined by the tactical and strategic priorities of the methodology.

How a methodology is defined and carried out is heavily influenced by the governance system. Essentially, the methodology must be determined so that it follows the constraints established by the governance system and the corresponding methods (processes) must be carried out in compliance with these constraints, as well as any additional constraints that may be further introduced by the methodology itself.

Governance and Management

Whereas a governance system establishes rules and constraints, it is not responsible for enforcing them or overseeing related activities to ensure compliance. Management refers to the system and resources responsible for day-to-day operations.

Within an IT environment, this basically pertains to the execution of activities. In relation to governance, a management system provides the hands-on means by which the

constraints and goals of the governance system are realized in the real world. Therefore, the management of a governance system represents a subset of the overall management responsibilities.

Management systems are assigned to and carried out by those with authority.

Methodology and Management

Management relates to methodology the same way it relates to governance. When building software programs according to a pre-defined methodology, a management system is used to ensure the proper execution of processes and project delivery lifecycle stages in compliance with the constraints of the methodology—and the constraints of the governance system.

Comparisons

The following list contains a series of sample distinctions to further help provide a clear separation between governance, methodology, and management:

- Governance establishes rules that control decision-making.
- Methodology establishes processes that comply to governance rules and may introduce additional rules.
- Management makes decisions according to governance rules.
- Governance does not dictate when or how to make a decision. It determines who should make the decision and establishes limits for that person or group.
- Methodology establishes processes that carry out specific types of decision logic that adhere to governance rules.
- Management is responsible for day-to-day operations and for ensuring that decisions made adhere to governance and methodology rules.
- Governance cannot replace management or methodology, nor can it compensate for poor management or poor (or inappropriate) methodology.
- Poorly defined and executed methodology can jeopardize the business goals associated with governance.
- Poor management can undermine a governance system and a methodology and will jeopardize associated business goals.
- Neither management nor methodology can replace governance, nor compensate for poor governance.

- A poor governance system inevitably inhibits the ability of a methodology to fulfill business automation requirement potential.
- A poor governance system inevitably inhibits the ability of management to make correct decisions.

As previously stated, while this book will make many references to management and methodology, it is primarily focused on governance.

STYLES OF GOVERNANCE

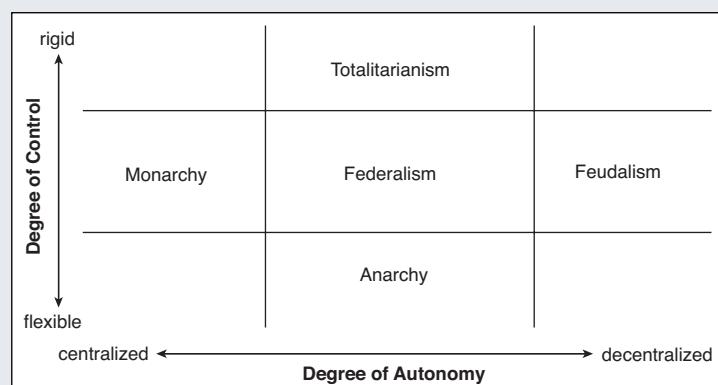
Governance must reflect and complement an organization's culture and structure. For example, when establishing suitable governance rules, considerations such as the following need to be raised:

- How much autonomy should each division, business unit, or department have?
- How much freedom should decision-makers have to delegate responsibilities to others?
- How much freedom should decision-makers have to use their own judgment when making decisions (as opposed to making decisions fully or partially based on pre-determined criteria)?

To determine what style of governance may be the best fit for a given organization, it can be helpful to refer to established forms of governance used historically in society. Figure 6.2 illustrates two dimensions that relate common governance styles.

Figure 6.2

The horizontal axis represents the degree of autonomy given to separate people or groups. The vertical axis represents the degree of control imposed on decision-makers.



Looking at one end of the horizontal spectrum, all decision-making is centralized, which is comparable to a monarchy. At the other end, each group establishes its own policies and procedures, similar to a feudal society. Many IT departments opt for a federated model, which permits the separation of the department into individual business units or cost centers, each of which is given a degree of independence while still maintaining a level of consistency. This helps reduce contention between fiefdoms.

When we study the vertical spectrum, we have a totalitarian type of regime whereby rigid policies dictate required actions, and decision-makers have little freedom to apply their own judgment. Too much rigidity can generate resentment and inhibit creativity in an organization. On the other hand, allowing flexible policies that provide only suggestive guidance leaves decision-makers with so much freedom that there is little chance of achieving meaningful consistency.

Good governance empowers people to do what's right for the business. Poor governance unnecessarily constrains or inhibits decisions, or fails to provide enough decision-making guidance. All governance—whether good or bad—places limits on the decisions and behaviors of the people being governed. It also prescribes consequences for those choosing not to abide. There is no single governance style that is correct for all organizations. Each must strive to find a balance between centralization and decentralization, between rigidity and flexibility, and between its existing culture and its ability to adapt to new approaches.

The Building Blocks of a Governance System

So far we've established that governance provides a systematic way for organizations to make decisions. Let's take a closer look at the primary building blocks that comprise a governance system:

- *precepts* define the rules that govern decision-making
- *people* assume roles and make decisions based on precepts
- *processes* coordinate people and precept-related decision-making activities
- *metrics* measure compliance to precepts

Note that these building blocks can be collectively or individually referred to as *governance controls*.

Precepts

A *precept* is an authoritative rule of action. Precepts are the essence of governance because they determine who has authority to make decisions, they establish constraints for those decisions, and they prescribe consequences for non-compliance.

Precepts codify decision-making rules using:

- *objectives* – broadly define a precept and establish its overarching responsibility, authority, and goals
- *policies* – define specific aspects of a precept and establish decision-making constraints and consequences
- *standards* – specify the mandatory formats, technologies, processes, actions, and metrics that people are required to use and carry out in order to implement one or more policies
- *guidelines* – are non-mandatory recommendations and best practices

NOTE

Within some IT communities, the term “policy” is commonly used instead of “precept” in relation to governance systems. However, as just explained, a policy can be just one aspect of a precept.

Also, even though a precept can contain standards, certain precepts themselves are considered standards. Therefore, it is important to not be confused when the precept name includes the word “standard” (such as Service Design Standard precept), and the precept itself further contains one or more standards that support corresponding precept policies.

People (Roles)

People (and groups of people) make decisions in accordance to and within the constraints stipulated by governance precepts. For a governance system to be successful, people must understand the intents and purposes of the precepts and they must understand and accept the responsibilities and authorities established by the precepts. Governance systems are therefore often closely associated with an organization’s incentive system. This allows the organization to foster a culture that supports and rewards good behavior, while also deterring and punishing poor behavior.

When exploring the involvement of people in relation to governance systems, it is further necessary to identify the role or roles they assume. Organizational roles position people (and groups) in relation to governance models and further affect the relevance of precept compliance and enforcement.

There are two ways that people can relate to precepts and processes: they can help author the precepts and processes and they can be dictated by their application. In this book, we explore both types of relationships.

Processes

A process is an organized representation of a series of activities. It is important to make a distinction between governance processes and other types of processes related to IT. Governance processes provide a means by which to control decisions, enforce policies, and take corrective action in support of the governance system. Other processes, such as those employed to carry out project delivery stages, can be heavily influenced by governance precepts, but are not specifically processes that are directly related to carrying out the governance system. Technically, any process is considered a management activity, but a governance system is dependent on governance processes to ensure compliance with its precepts.

An organization is likely to use a variety of processes to support its precepts. Some may be automated, while others require human effort. Automated processes can help coordinate tasks (such as steps required to collect data for approvals), but can still rely on people to make important decisions (such as making the actual approvals based on the presented data).

Metrics

Metrics provide information that can be used to measure and verify compliance with precepts. The use of metrics increases visibility into the progress and effectiveness of the governance system. By analyzing metrics, we gain insight into the efficacy of governance rules and we can further discover whether particular precepts or processes are too onerous or unreasonable. Metrics also measure trends, such as the number of violations and requests for waivers. A large number of waiver requests may indicate that a given precept might not be appropriate or effective.

Governance and SOA

An organization establishes governance to mitigate risk and to help advance its strategy, goals, and priorities. When the organization invests in an SOA initiative, it expects to gain benefits worth more than the cost of the investment. This return on investment is measured in terms of business outcomes, and, presumably, those outcomes reflect the organization's strategy, goals, and priorities. Therefore, the primary business goal for SOA governance is to ensure that an SOA initiative achieves its targeted business outcome.

An SOA governance system is the meta-decision system that an organization puts in place to control and constrain decision-making responsibilities related to the adoption and application of service-orientation. There are many practices, considerations, models, and frameworks that can comprise a meta-decision system suitable for SOA governance, all of which are explored throughout this book. The foundation of an SOA governance system resides within an SOA Governance Program Office responsible for creating and administering an SOA governance program that encompasses and defines necessary SOA governance models and the tasks required to realize and sustain these models.

NOTE

The term “SOA Governance Program Office” is intentionally capitalized as it represents the official name of an IT department. The term “SOA governance program” is not capitalized, as it refers to a type of program that is commonly assigned its own unique name.

SUMMARY OF KEY POINTS

- There are clear distinctions between governance, methodology, and management.
 - The building blocks of a governance system are precepts, people, processes, and metrics.
 - The fundamental steps to laying the foundation for an SOA governance system are to create an SOA Governance Program Office that creates and administers an SOA governance program.
-

6.2 The SOA Governance Program Office (SGPO)

NOTE

For simplicity's sake this chapter frequently uses the acronym "SGPO" for the "SOA Governance Program Office." This is not an industry-standard acronym, nor is the book proposing it as such. It is an acronym used solely to simplify content by avoiding repeatedly spelling out this term.

The first step in any SOA governance effort is to establish a group (or department) that assumes the responsibility of defining and administering the various parts of an SOA governance system. This group forms the SOA Governance Program Office (SGPO), an organizational entity that is commonly comprised of trained SOA Governance Specialists, Enterprise Architects, and other types of IT decision-makers. The SGPO is given the authority to define and enforce the on-going activities and rules associated with SOA governance.

A primary responsibility of the SGPO is to author a series of formal precepts. In some cases, the SGPO may need to request amendments to existing IT governance precepts to accommodate the distinct needs of SOA projects, as the SGPO needs to avoid inadvertently defining conflicting precepts.

In general, SOA governance precepts are more balanced and more easily accepted when those who are governed have a voice. The SGPO may therefore need to solicit input from major stakeholders, including IT and business managers, senior IT staff, and even the legal department. Those contributing should have an opportunity to comment on pending precepts, propose amendments, and recommend new precepts. However, just because the SGPO solicits input does not imply that it is relinquishing its authority to establish the necessary SOA governance precepts.

Following are some basic guidelines for incorporating the SGPO into an IT environment:

- The SGPO must have the responsibility and authority to develop and manage the SOA governance system, and other teams must accept the SGPO's authority.
- The SGPO must ensure that the SOA governance system aligns with the organization's incentive and disciplinary systems.
- The SGPO must develop collaborative working relationships with other governance teams whose responsibilities intersect with those of the SGPO.

- The SGPO must ensure that its precepts align with other governance systems (Figure 6.3) within the company, or they must work with the other governance program offices to amend the conflicting precepts.
- The SGPO must have access to communication channels to disseminate information about the governance precepts and to provide training to people affected by them.

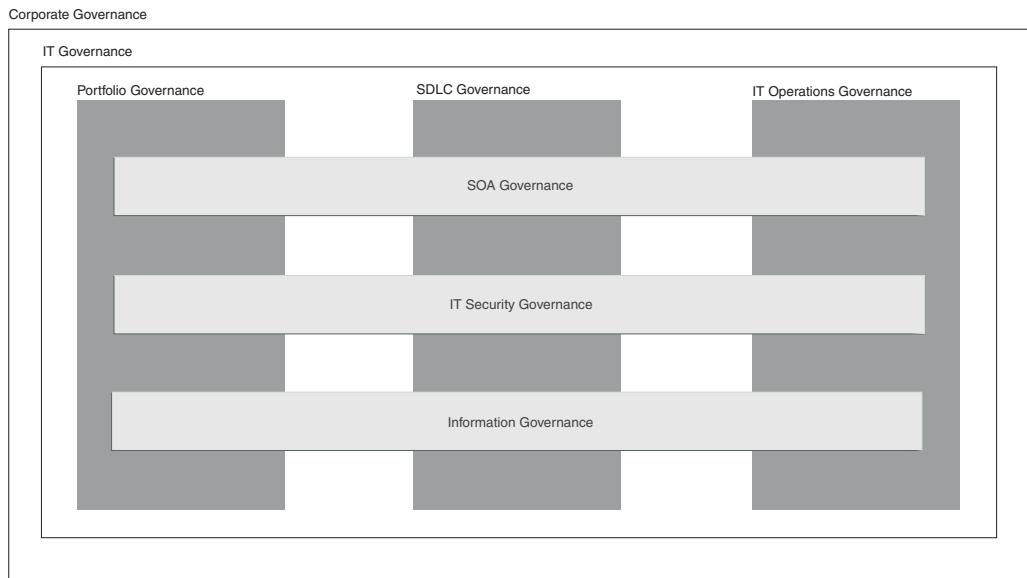


Figure 6.3

SOA governance must be defined through a program that can harmoniously co-exist alongside other IT governance programs.

What's of critical importance is that an appropriate scope be established for the SGPO. There are two primary factors that determine this scope: the reach of the SGPO within the overall IT enterprise and the areas of responsibility assumed by the SGPO within whatever domain it operates.

6.3 SGPO Jurisdiction Models

As explained in Chapter 3, a given IT enterprise can have one or more service inventories. Each service inventory represents a collection of independently standardized and governed services. When an IT enterprise has multiple service inventories, each is (ideally) associated with a well-defined domain, such as a line of business. In this case, service inventories are further qualified with the word “domain.”

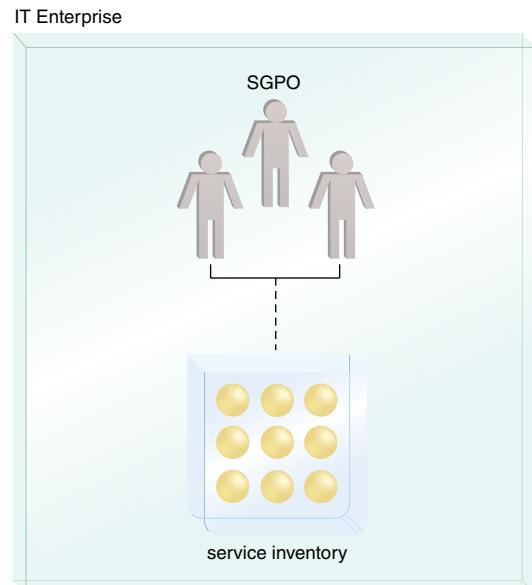
Depending on whether domain service inventories are being used and depending on how cooperative relations are between different service inventory owners, there may or may not be the opportunity to have one SGPO assume responsibility for multiple domain service inventories. As a result, different jurisdiction models exist, as follows:

Centralized Enterprise SGPO

If a single enterprise service inventory has been established, then it is generally expected that SOA governance responsibilities will be assigned to a single SGPO that oversees SOA governance on behalf of the entire IT enterprise.

Figure 6.4

A single SGPO responsible for the enterprise service inventory.



Centralized Domain SGPO

Even though individual domain service inventories can be independently standardized, managed, and owned, with enough cooperation between the owners, the IT department may be able to establish a single, enterprise-wide SGPO that subjects all service inventories to a common SOA governance system.

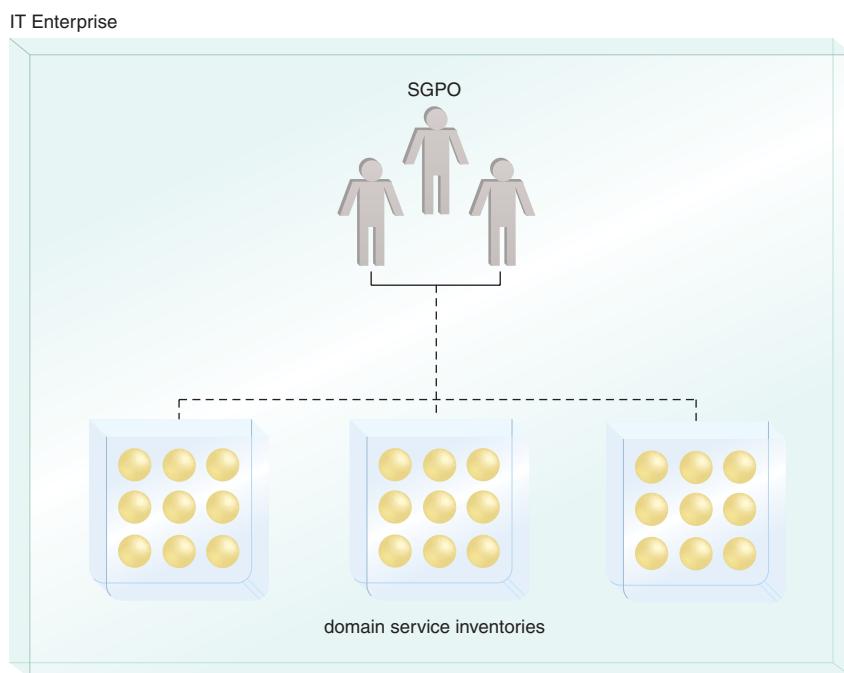


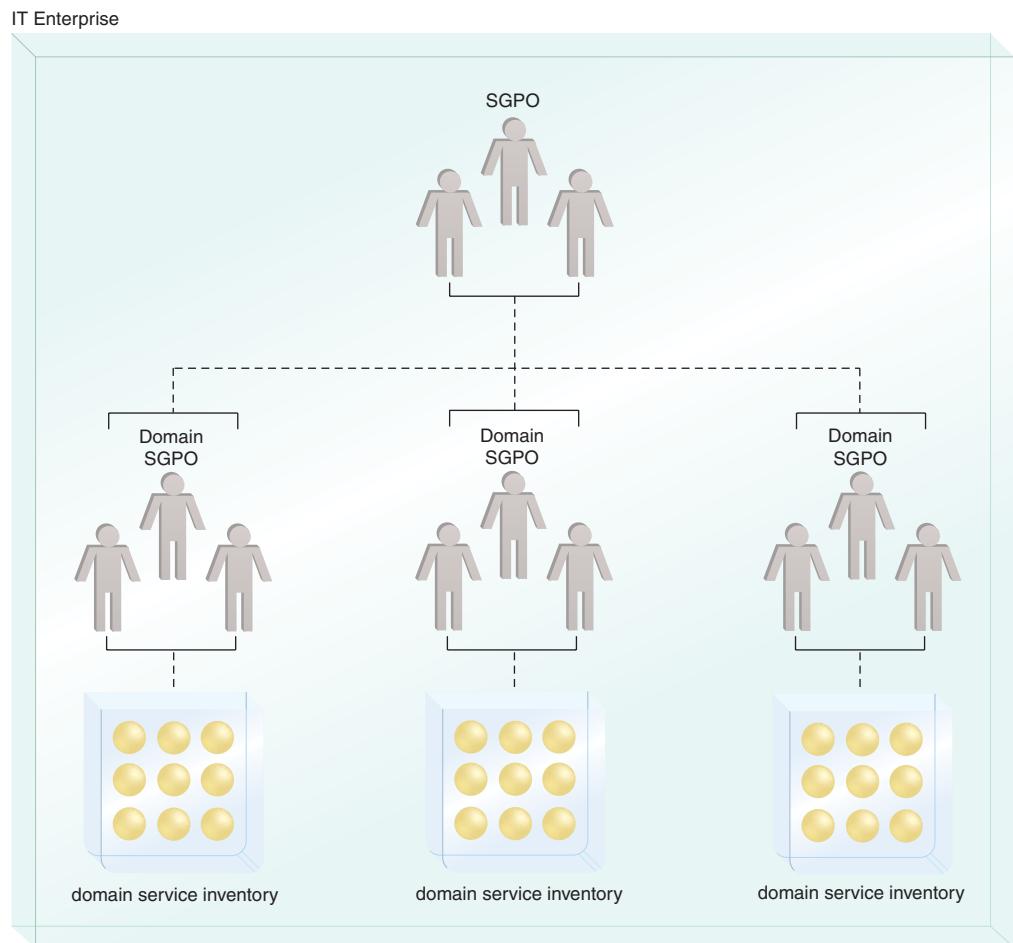
Figure 6.5

A single SGPO responsible for multiple domain service inventories.

Alternatively, different SOA governance programs can be created for each or select domain service inventories. With this model, separate programs can still be defined and maintained by the same central SGPO. The primary benefit of doing so is to maintain consistency and enterprise-wide alignment of how SOA governance programs are created and carried out, despite the fact that the respective SOA governance systems vary.

Federated Domain SGPOs

In this model, a central overarching SGPO exists in addition to individual SGPOs, each responsible for a separate domain service inventory. The domain SGPOs carry out individual SOA governance programs; however, these programs are required to comply to a set of conventions and standards defined by a single parent SGPO. The intent of this model is to strike a balance between domain-level independence and enterprise-wide consistency.

**Figure 6.6**

Multiple domain SGPOs are further “governed” by a central overarching SGPO.

Independent Domain SGPOs

Each domain service inventory has its own SGPO, which has full governance authority and jurisdiction over that domain. With the absence of a centralized SGPO presence, independent domain-level SGPOs have complete freedom to define and execute respective SOA governance programs.

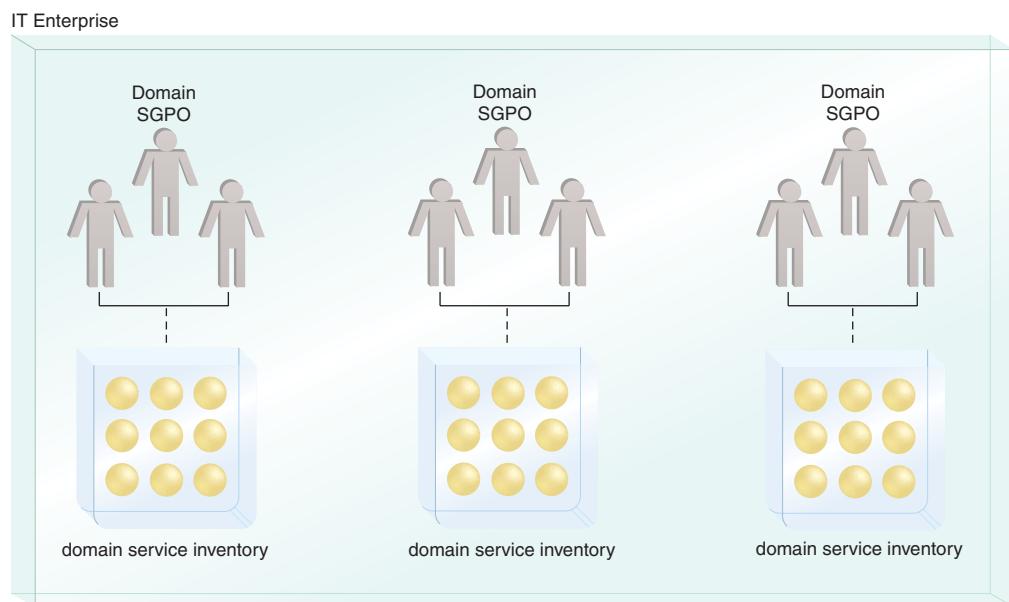


Figure 6.7

Multiple domain SGPOs independently govern multiple domain service inventories.

SUMMARY OF KEY POINTS

- The SGPO is an organizational entity responsible for defining and administering the SOA governance program.
 - The SGPO needs to be carefully positioned within the overall IT department to ensure alignment with existing governance groups and programs.
 - Different SGPO jurisdiction models can be considered, depending on the SOA adoption approach taken by an organization.
-

6.4 The SOA Governance Program

The SGPO exists to create and maintain an *SOA governance program*. This program encompasses the SOA governance system and all associated responsibilities for planning, implementing, and evolving this system. The best way to distinguish the program from the system is to view the SOA governance system as a set of formal precepts, roles, processes, metrics, and any associated models. The SOA governance program is dedicated to establishing and evolving the SOA governance system and therefore further provides real-world planning and implementation considerations, such as project plans, budgets, schedules, milestones, and further deliverables that map the SOA governance system to other parts of the existing IT enterprise (including already established IT governance systems).

The task of realizing an SOA governance program can be divided into three basic steps:

1. Assessing the Enterprise (or Domain)
2. Planning and Building the SOA Governance Program
3. Running the SOA Governance Program

Step 1: Assessing the Enterprise (or Domain)

Before creating appropriate precepts and formalizing the overall SOA governance system, the SGPO must first evaluate specific aspects of the current organizational state of the IT enterprise or whatever domain thereof for which that SOA adoption is being planned. This assessment may be limited to the domain in which the SGPO operates, but often also encompasses broader, organization-wide considerations that apply to most or all domains.

The assessment generally focuses on several specific areas:

- Current Governance Practices and Management Styles
- SOA Initiative Maturity
- Current Organizational Model
- Current and Planned Balance of On-Premise and Cloud-based IT Resources

Current Governance Practices and Management Styles

The organization's existing governance practices and management styles need to be studied to determine how best to introduce SOA governance-related processes and precepts. As previously described, no one governance model is suitable for every organization. A successful SOA governance program must take into account the organization's culture and management preferences.

Common issues that need to be addressed include:

- Are decisions tightly controlled by a central authority or widely delegated?
- Do the various groups within the organization collaborate or do they typically work autonomously?
- How do other governance program offices in the company work?
- How well does the organization articulate and disseminate governance precepts?
- How rigorously do people within the organization adhere to standard practices and processes?
- How much flexibility do managers and project leaders have in adapting to processes to meet the needs of a specific project?
- How much flexibility does management have to establish or modify incentive systems?

Concrete, well-researched answers to these questions can significantly influence an SOA governance program in that they can identify both strengths and weaknesses in relation to the types of governance and management practices required to see through a successful SOA initiative. This, in turn, helps determine the nature of precepts required and to what extent the existing IT culture will be impacted by the SOA governance system.

SOA Initiative Maturity

Ideally, an SOA governance program is established prior to the launch of an SOA initiative. However, in situations where existing SOA projects or activities are already underway, a further analysis of their progress and maturity is required to ensure that the introduction of the SOA governance program ends up supporting and aligning these efforts with overarching strategic goals. The SGPO may also need to spend time assessing existing SOA initiatives in relation to an IT department's readiness for SOA governance.

NOTE

Visit www.soaspes.com for a list of industry maturity models relevant to the adoption of service-orientation and SOA.

Current Organizational Model

An organizational model defines roles and responsibilities within an organization. A given IT department will have a distinct organizational model that usually establishes a hierarchy with levels of authority. The SGPO must assess existing roles and responsibilities in order to identify how new roles and responsibilities specific to SOA governance will affect the organizational model.

Current and Planned Balance of On-Premise and Cloud-based IT Resources

In order to take an appropriate range of considerations into account when authoring SOA governance precepts and supporting processes, the SGPO needs to have a clear understanding of what cloud-based IT resources relevant to the SOA project currently exist, and to what extent the organization is planning to explore or proceed with cloud-based deployment of services and/or related IT resources. These considerations usually lead to additional standards, additional factors that apply to review processes, and additional organizational roles and skill-sets required for the definition of precepts and processes.

Step 2: Planning and Building the SOA Governance Program

After assessing the organization, the SGPO can get to work on actually planning and creating a concrete program for SOA governance. As previously established, the SOA governance program encompasses the SOA governance system and further provides supporting components to help establish and maintain this system.

To identify the primary components of an SOA governance program, we therefore begin by revisiting the precepts, people, and processes that are part of a governance system.

SOA Governance Precepts

The assessment completed in the previous stage is intended primarily to identify the aspects of a current or planned SOA initiative that pose the most risk and have the most urgent need for structured governance.

The following precepts are described individually in Chapters 7 to 12, where they are further associated with project lifecycle stages, processes, and organizational roles:

- Service Profile Standards (Chapter 7)
- SOA Governance Technology Standards (Chapter 7)
- Preferred Adoption Scope Definition (Chapter 7)
- Organizational Maturity Criteria Definition (Chapter 7)
- Standardized Funding Model (Chapter 7)
- Service Inventory Scope Definition (Chapter 8)
- Service and Capability Candidate Naming Standards (Chapter 8)
- Service Normalization (Chapter 8)
- Service Candidate Versioning Standards (Chapter 8)
- Schema Design Standards (Chapter 9)
- Service Contract Design Standards (Chapter 9)
- Service-Orientation Contract Design Standards (Chapter 9)
- SLA Template (Chapter 9)
- Service Logic Design Standards (Chapter 9)
- Service-Orientation Architecture Design Standards (Chapter 9)
- Service Logic Programming Standards (Chapter 9)
- Custom Development Technology Standards (Chapter 9)
- Testing Tool Standards (Chapter 10)
- Testing Parameter Standards (Chapter 10)
- Service Testing Standards (Chapter 10)
- Cloud Integration Testing Standards (Chapter 10)
- Test Data Usage Guidelines (Chapter 10)
- Production Deployment and Maintenance Standards (Chapter 10)
- Runtime Service Usage Thresholds (Chapter 11)

- Service Vitality Triggers (Chapter 11)
- Centralized Service Registry (Chapter 11)
- Service Versioning Strategy (Chapter 11)
- SLA Versioning Rules (Chapter 11)
- Service Retirement Notification (Chapter 11)
- Enterprise Business Dictionary/Domain Business Dictionary (Chapter 12)
- Service Metadata Standards (Chapter 12)
- Enterprise Ontology/Domain Ontology (Chapter 12)
- Business Policy Standards (Chapter 12)
- Operational Policy Standards (Chapter 12)
- Policy Centralization (Chapter 12)

It is important to document the reasoning behind each precept and define the circumstances in which it does or does not apply. Precepts need to be codified with clarifying policies and standards and consequences for non-compliance need to be further established. Also, supporting guidelines and compliance metrics are required. Where appropriate, conditions that might warrant a waiver need to be identified and a separate precept for allowing or denying waivers may further be required.

SOA Governance Processes

Depending on the size of the SGPO, internal processes may be required to coordinate activities within the group running the office. Governance process definition is another area of focus for the SOA governance program.

The following processes are covered in Chapters 7 to 12, where they are mapped to project lifecycle stages, precepts, and organizational roles:

- Organizational Governance Maturity Assessment (Chapter 7)
- Adoption Impact Analysis (Chapter 7)
- Adoption Risk Assessment (Chapter 7)
- Business Requirements Prioritization (Chapter 8)
- Service Candidate Review (Chapter 8)

- Service Contract Design Review (Chapter 9)
- Service Contract Registration (Chapter 9)
- Service Access Control (Chapter 9)
- Service Logic Design Review (Chapter 9)
- Legal Data Audit (Chapter 9)
- Service Logic Code Review (Chapter 9)
- Service Test Results Review (Chapter 10)
- Service Certification Review (Chapter 10)
- Service Maintenance Review (Chapter 10)
- Service Vitality Review (Chapter 11)
- Service Registry Access Control (Chapter 11)
- Service Registry Record Review (Chapter 11)
- Service Discovery (Chapter 11)
- Shared Service Usage Request (Chapter 11)
- Shared Service Modification Request (Chapter 11)
- Service Versioning (Chapter 11)
- Service Retirement (Chapter 11)
- Data Quality Review (Chapter 12)
- Communications Quality Review (Chapter 12)
- Information Alignment Audit (Chapter 12)
- Policy Conflict Audit (Chapter 12)

You may have noticed how several of these processes end with “review.” Many SOA governance processes are designed specifically to support and enforce compliance to precepts, and therefore are carried out subsequent to other project delivery tasks as a formal review.

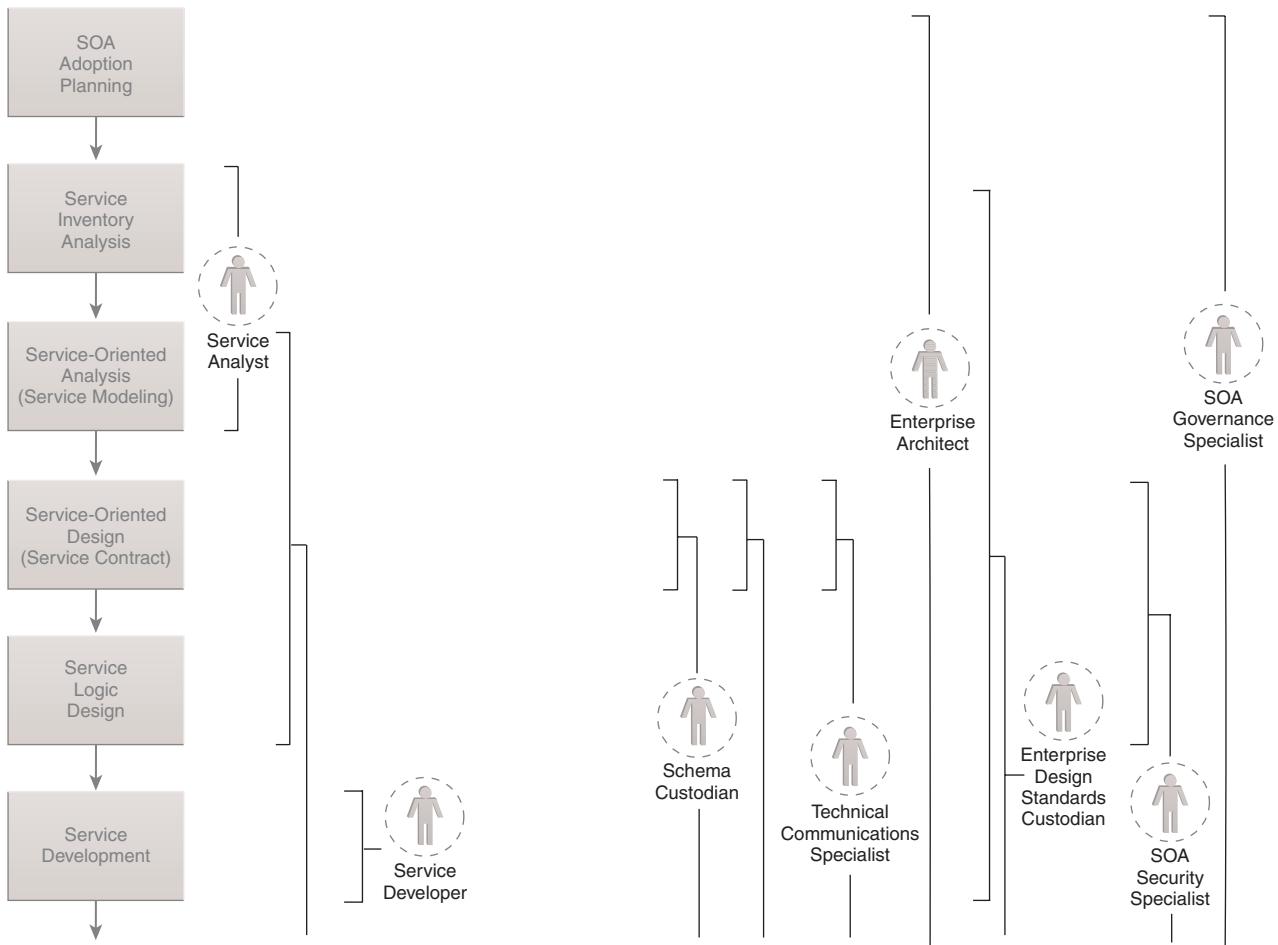
SOA Governance Roles

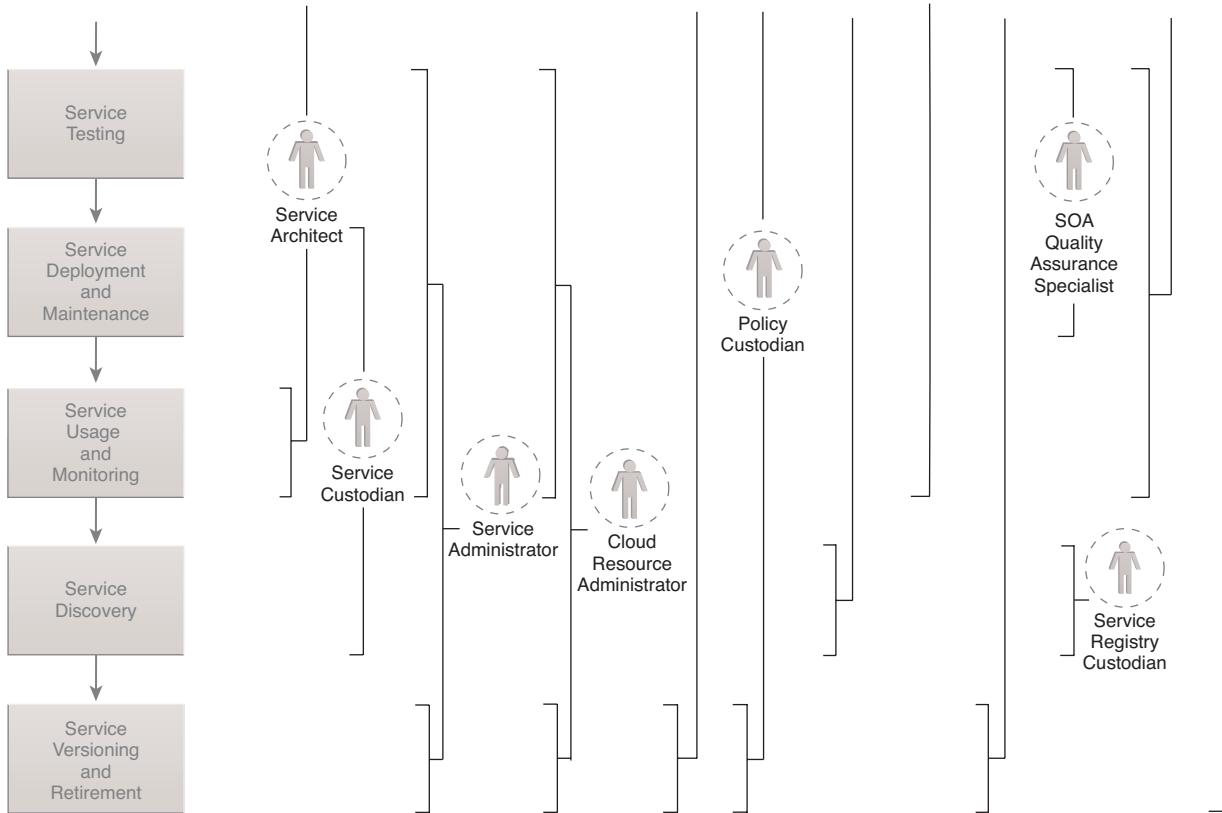
Organizational roles associated with SOA initiatives are of great interest to the SGPO because the various project stages for which governance precepts and processes can be defined will involve these roles in a governance capacity.

The following organizational roles were introduced in Chapter 5 and are further explored in Chapters 7 to 12, where they are associated with project lifecycle stages and SOA governance precepts and processes:

- Service Analyst
- Service Architect
- Service Developer
- Service Custodian
- Service Administrator
- Cloud Resource Administrator
- Schema Custodian
- Policy Custodian
- Service Registry Custodian
- Technical Communications Specialist
- Enterprise Architect
- Enterprise Design Standards Custodian (and Auditor)
- SOA Quality Assurance Specialist
- SOA Security Specialist
- SOA Governance Specialist

Figure 6.8 provides an overview of how these roles commonly map to SOA project life-cycle stages.



**Figure 6.8**

Each role can be involved in governance activities pertaining to multiple SOA project stages. Appendix B further provides master reference diagrams that illustrate the cross-project stage relationships of these roles with precepts and processes.

Additional Components

As previously stated, the scope of the SOA governance program goes beyond the definition of the SOA governance system. Some of the areas that the program will likely need to further address in support of pre-defined precepts and processes include:

- *SOA Governance Tools* – Products and technologies that enable the automation of SOA governance processes or that can monitor and collect relevant statistical data need to be identified and chosen in order to establish a suitable SOA governance infrastructure.
- *SOA Governance Roadmap* – Also referred to as the SOA Governance Program Project Plan, this document establishes the timeline, resources, budget, and other real-world considerations required to actually realize the goals of the SGPO and, more specifically, a specific SOA governance program.

There can be many more parts and extensions to an SOA governance program specific to the needs of a given IT department and its SOA project goals.

Step 3: Running the SOA Governance Program (Best Practices and Common Pitfalls)

The SOA governance program is a living entity that requires continuous maintenance. Over time, and in response to real-world issues and challenges, the SOA governance program will naturally evolve as precepts, roles, and processes are refined or added to the overall SOA governance system.

This section contains a series of best practices that provide guidance for successfully running an SOA governance program, as well as a set of common pitfalls that warn against factors and circumstances that can inhibit the adoption and evolution of the program.

Collect the Right Metrics and Have the Right People Use Them

Metrics, the fourth primary building block of a governance system, represent a vital element in the on-going operation of the SOA governance program. Having the tools and processes to consistently collect and disseminate key metrics is just as important as having the right individuals and groups assigned the responsibility to interpret and make decisions based on the reported metrics.

Provide Transparency and Foster Collaboration

Depending on its scope, an SOA governance program can affect a wide range of departments, groups, and individuals. Instead of creating the program in isolation, its development should be an open process, accessible for review and involvement to others within the IT department. Not only will this generate goodwill among those less enthusiastic about upcoming SOA adoption initiatives, but it will also allow people to voice concerns and provide suggestions. This type of feedback can help improve the SOA governance system, while also easing its eventual implementation.

Ensure Consistency and Reliability

SOA governance precepts need to be consistently enforced and SOA governance processes need to be consistently carried out. Providing a reliable means of managing and maintaining the SOA governance system is the foremost responsibility of the SGPO and depends heavily on the quality and detail with which the SOA governance program has been developed.

Besides human incompetence and poor SOA governance program definition, another reason this best practice may not be followed is an unexpected withdrawal of funding allocated to the SGPO. Should this occur, it is preferable to downsize the scope of the SOA governance program instead of trying to continue carrying out SOA governance activities without the necessary resources to ensure consistency and reliability.

Compliance and Incentives

An SOA governance system will introduce precepts that will sometimes restrict certain tasks that IT project team members have traditionally been free to complete by using their own judgment. At the same time, precepts also help make critical decisions for IT professionals that can ease their responsibilities while also guaranteeing consistency across services and service-oriented solutions. It is important that project teams embrace SOA governance precepts and processes and that they clearly understand how and why new types of compliance are required, while also fully acknowledging that their judgment and freedom in other areas are still required and relied upon.

Furthermore, offering formal incentives for regularly supporting precepts can go a long way to fostering consistent adherence. Because people will generally do that for which they are most rewarded, an absence of incentives can encourage them to violate or ignore SOA governance precepts. When this happens, something generally needs to change: the incentive, the precept, or the people.

Education and Communication

SOA governance systems can impose precepts more restrictive than traditional IT governance systems. Furthermore, some organizations can find it difficult to fully mandate the adoption of and compliance to SOA governance precepts. Even when compliance is required, in some IT cultures, groups or individuals can still choose to “rebel” by intentionally disregarding precepts because they are considered too burdensome.

Regardless of whether compliance to SOA governance precepts is voluntary or mandatory, it is critical that everyone affected fully understand why these precepts exist and how their compliance ultimately results in tangible benefits. Furthermore, it can be helpful to specifically address the common question: “What’s in it for me?” Fostering a true understanding of how support for the SOA governance system can result in personal benefit will further help unify IT project teams and personnel.

For this purpose, the SGPO must put together an education and communications program. This program must begin by establishing SOA terminology, concepts, and practices using a common vocabulary that all project team members can understand. It must then introduce the SOA governance system and impress its virtues.

Common Pitfalls

From the many failed and successful SOA adoption initiatives has emerged a set of common pitfalls that pertain directly to establishing and running an SOA governance program:

- *Lack of Recognized Authority* – The SGPO must be endowed with the responsibility and authority to develop and execute the SOA governance program. For this to happen, other IT departments and project teams must accept that authority. When the SGPO’s authority is ignored or not recognized, there needs to be recourse. If the lack of recognition persists, there need to be consequences for those who refuse to provide support.
- *Misalignment with IT Governance* – An SOA governance system must be consistent with and supportive of existing corporate and IT governance systems. If other IT governance precepts and processes are not taken into consideration, the SOA governance system can become inadvertently misaligned. This will result in conflicts and can further introduce risks to the IT department as a whole.
- *Overestimating or Underestimating Cloud Computing Factors* – There are various ways that cloud platforms and technologies can be made part of the planned SOA project. An organization may have or may plan to establish a private cloud comprised

of standardized IT resources that require distinct administration processes, or it may be moving IT resources to a public cloud that imposes non-compliant requirements that may require even more distinct administration approaches. Either way, it is important for the SGPO to be open and flexible regarding these possibilities and—if cloud deployment is a possibility—to fully understand the consequences of having some or all services or IT resources of a given project deployed in cloud environments.

- *Impractical or Overly Formal Processes* – SOA governance processes are intended to help enforce and organize the application of precepts. Sometimes it can be tempting to create highly structured and detailed processes that cover all possible bases. Although such processes may be thorough, they can be too burdensome, onerous, or time consuming to carry out consistently in the real world. When designing SOA governance processes, consider the impact of the process on the project lifecycle and timeline and investigate any opportunity to streamline and automate parts of the process. Tools that integrate the governance process directly with development or administration platforms may further be helpful in allowing developers and administrators to efficiently identify and fix compliance issues.
- *Poor Documentation* – SOA governance precepts should be well-documented and disseminated. Many precepts require human interpretation, which means that people in the trenches will need to clearly understand how and when to apply them. Sometimes members of the SGPO take the formality of an SOA governance system too seriously. As a result, precepts and processes can be documented using overly academic or technical language. This can make the documents difficult to fully understand and, at times, inaccessible to some project team members.
- *Overspending on SOA Governance Tools* – SOA vendors have developed highly sophisticated administration and management tools (commonly labeled as “governance” products) with various design and runtime features. While powerful, these tools sometimes provide functionality that is not needed or not suitable for an organization’s specific governance requirements. Further, these tools can be very expensive, especially in larger IT enterprises. Therefore, it is often worth waiting to invest in a full-blown SOA governance infrastructure until an SOA governance program has matured to the extent that the actual design and runtime automation requirements can be identified and well defined. Otherwise, over-spending or mis-spending on governance tools and technology can put a significant dent in an SOA initiative’s overall ROI and further limit funds that may have been better allocated to supporting the SGPO in other areas.

SUMMARY OF KEY POINTS

- An SOA governance program encompasses the models that comprise an SOA governance system and further provides actionable artifacts that determine how the system will be established and maintained.
 - A basic framework for an SOA governance program consists of three primary parts that address the assessment of the current organizational state, the planning and building of the program, as well as its evolutionary operation.
-

A black and white photograph showing a close-up, low-angle view of a modern building's exterior. The facade is composed of numerous vertical, ribbed, metallic panels that curve outwards. The lighting creates strong highlights and shadows on the metallic surfaces, emphasizing the texture and depth of the curves.

Part II

Project Governance

Chapter 7: Governing SOA Projects

Chapter 8: Governing Service Analysis Stages

Chapter 9: Governing Service Design and Development Stages

Chapter 10: Governing Service Testing and Deployment Stages

Chapter 11: Governing Service Usage, Discovery, and Versioning Stages

This page intentionally left blank

Chapter 7



Governing SOA Projects

7.1 Overview

7.2 General Governance Controls

7.3 Governing SOA Adoption Planning

SOA PRINCIPLES & PATTERNS REFERENCED IN THIS CHAPTER

- Brokered Authentication [496]
- Data Confidentiality [512]
- Data Origin Authentication [515]
- Direct Authentication [518]
- Entity Abstraction [524]
- Exception Shielding [526]
- Message Screening [534]
- Process Abstraction [544]
- Service Layers [561]
- Service Perimeter Guard [564]
- Trusted Subsystem [571]
- Utility Abstraction [573]

The foremost goal of an SOA governance program is to establish precepts for the effective, end-to-end governance of SOA project lifecycle stages. The chapters in this part of the book step through these stages (Figure 7.1) and highlight relevant governance considerations, including proposed precepts and supporting processes, roles, and metrics. It is best to consider the next six chapters as reference material, meaning that it is not necessary to read the chapters or the sections within chapters in sequence. You may want to focus on the project stages or organizational roles that are most relevant to you.

7.1 Overview

As explained in the previous chapter, the SOA Governance Program Office is not responsible for the definition of SOA project stages. Its concern is establishing entry and exit criteria for each stage. An effective approach to achieving this is to ensure that the quality and completeness of outputs from each individual lifecycle stage are reviewed and approved before proceeding to the next stage. For this reason, several review processes based on criteria established by related precepts are introduced over the next set of chapters. Adherence to these precepts is important. The longer it takes for an error or omission to be noticed, the greater its eventual impact and the cost of its correction.

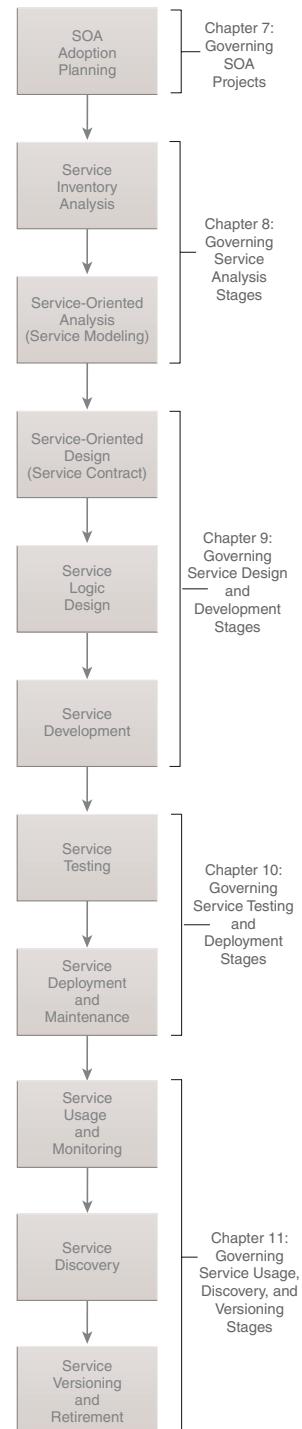


Figure 7.1

The SOA project lifecycle stages. Each of these stages was introduced in Chapter 5. The chapters in this part of the book focus solely on SOA governance controls that pertain to these stages.

Precepts, Processes, and People (Roles) Sections

The topic areas covered by Chapters 7 to 11 in Part II (as well as Chapter 12 in Part III) are further sub-divided into sections that correlate with the fundamental building blocks of a governance system:

- *Precepts* – Common precepts pertaining to an SOA project stage or an overarching governance concern are documented. A precept can be associated with processes and roles. In some special cases, a precept can be associated with another precept.
- *Processes* – Common processes used to apply and realize the goals of precepts are documented wherever appropriate. A process can be associated with precepts and roles.
- *People (Roles)* – Common organizational roles (as first described in Chapter 5) are identified and described, as they relate to covered precepts and/or processes. This section highlights roles involved in the creation of precepts and processes, as well as those regulated by them.

The mapping provided in this book between precepts, processes, and roles is based on proven associations as part of a generic SOA governance program structure common in the industry. These mappings are recommended, not required. Custom variations tailored to fulfill the specific needs of the SOA project and the organization as a whole are encouraged.

Note that some precepts can be associated with processes from different project stages (and therefore described in different chapters), and vice versa. In this case, the referenced precept or process is further qualified with its project stage name and chapter number. However, due to the fact that most organizational roles already span project stages, the roles described individually in the upcoming chapters make no reference to precepts or processes outside of the project stage section in which the role is being described.

For example, the Adoption Impact Analysis process introduced in the *Service Inventory Analysis* section later in this chapter includes a reference to the Service Inventory Scope Definition precept covered in Chapter 8. Therefore, the reference is further qualified, as follows:

Service Inventory Scope Definition (Service Inventory Analysis, Chapter 8)

The *People (Roles)* sections in the upcoming chapters are limited to the mapping of roles to precepts and processes for a given project stage. Appendix B provides a series of master reference diagrams that show cross-project stage relationships of roles to precepts

and processes. Note also that mapping is provided only for the primary roles described in Chapter 5. Roles that are prefixed with “Other:” are provided for reference purposes but are not always mapped to precepts and processes.

HOW CASE STUDIES ARE USED IN PART II

Chapters 7, 8, 9, 10, and 11 are sub-divided into sections appended with associated case study examples. You may therefore want to revisit the case study background provided in Chapter 2 prior to reading these sections.

Many of the case study examples incorporate SOA governance precepts. Sometimes, precepts are further augmented to demonstrate how they can be customized and to highlight the fact that customization of the precepts and processes in this book is common and recommended.

7.2 General Governance Controls

Precepts

Before we look at precepts that are specific to individual SOA project stages, it is worth highlighting those that can apply to multiple stages or to the project as a whole.

Service Profile Standards

Services within a given service inventory need to be consistently documented, from when they are first conceived through all subsequent lifecycle stages, until they are eventually retired. Consistency in the type of information recorded about a service and in the format in which this information is organized is vital to effective service governance, especially considering how many different individuals and groups will make decisions based on this information.

The service profile described in Chapter 5 establishes a common format from which a custom standardized service profile can be derived for all services within a service inventory.

The generic service profile is organized into the following parts:

- Service-Level Profile Structure
- Capability Profile Structure

The various fields provided for each structure establish a flexible profile format in which different parts of the profile can be owned and maintained by different custodians and others involved with pre and post-deployment service lifecycle stages. However, governance regulations can limit this flexibility as required. For example, it may be necessary to restrict the maintenance of the service profile to a sole Service Custodian.

Service Information Precepts

Service information primarily represents business-centric data to which clear meaning and context has been assigned.

The following service information precepts are covered in Chapter 12:

- Business Dictionary
- Service Metadata Standards
- Ontology

Furthermore, the following supporting processes are described:

- Data Quality Review
- Communications Quality Review
- Information Alignment Audit

The use of service information has relevance to a number of stages, especially those that pertain to analysis (in that it assists with the identification of services and the definition of their functional boundaries).

Service Policy Precepts

As with service information precepts, regulations that govern the use of service policies are not stage-specific.

The following service policy precepts are also covered in Chapter 12:

- Business Policy Standards
- Operational Policy Standards
- Policy Centralization

These precepts tie into the Policy Conflict Audit process and they are further associated with service information precepts via the aforementioned Information Alignment Audit process.

Logical Domain Precepts

For many of the precepts that regulate the development, usage, and evolution of different services, there is the opportunity to further establish rules and guidelines (and perhaps even separate precepts) for services based on different types of service models.

Service models (explained in Chapter 3) form the basis of logical service layers within a given service inventory (Figure 7.2). Each service layer represents a type of service, classified by the nature of its functionality.

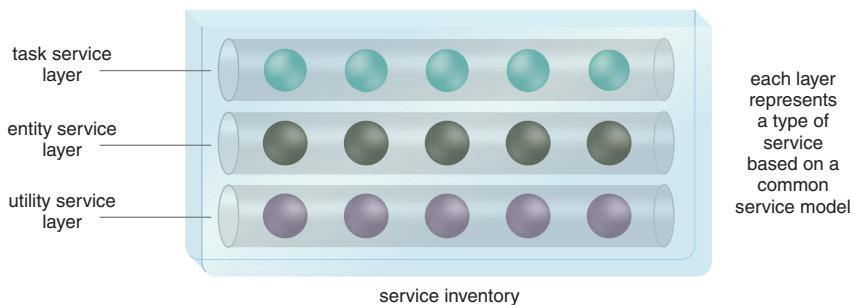


Figure 7.2

A service inventory with services organized into the three common types of service layers: utility, entity, and task, each based on a correspondingly named service model.

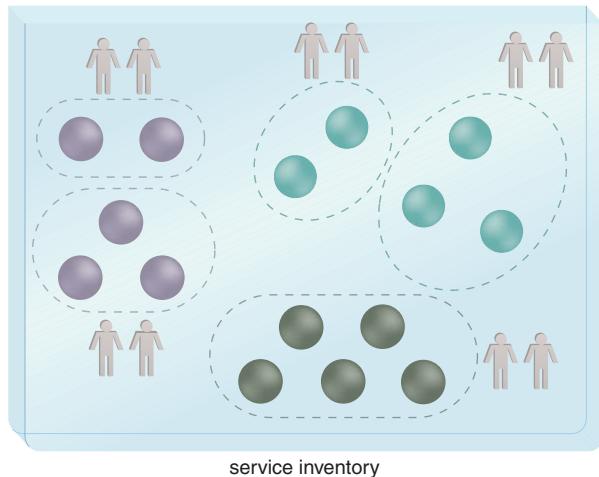
Different service layers can be governed in different ways when the governance precept in question pertains to differences in the functional context of the services (Figure 7.3). These service layer or service model-specific factors and considerations can apply to services when they are first conceptualized as part of early analysis stages, right through to when they are actively being used.

SOA PRINCIPLES & PATTERNS

The referenced service layers are based on the Service Layers [561] pattern, and the types of common service layers relate to the Utility Abstraction [573], Entity Abstraction [524], and Process Abstraction [544] patterns.

Figure 7.3

Each set of services is being governed by different people or groups, based on the nature of their functionality.



Security Control Precepts

Throughout any of the project lifecycle stages covered in Chapters 7 to 11 there can be the need to create specific precepts to address unique security requirements and concerns. These precepts can vary significantly in that they can establish governance for a range of security controls for areas such as:

- Trust
- Claims
- Tokens
- Identification
- Authentication
- Authorization
- Confidentiality
- Integrity
- Non-Repudiation

Security control precepts can help standardize and position the usage of common SOA-related security technologies and mechanisms, including:

- Transport-Layer Security
- Message-Layer Security

- Security Policies
- Security Token Actions
- Trust Brokering
- Security Sessions
- Encryption
- Hashing
- Digital Signatures
- Identity and Access Management (IAM)
- Public Key Infrastructure (PKI)
- Digital Certificates
- Certificate Authority (CA)
- Single Sign-On
- Cloud-Based Security Groups (common to cloud-based architectures)
- Hardened Virtual Server Images (common to cloud-based architectures)

Similarly, security control precepts can further help identify approved industry security standards and further standardize their usage within the overall technology architecture.

Examples of common SOA-related security industry standards include:

- XML-Encryption
- XML-Signature
- Canonical XML
- Decryption Transform for XML Signature
- Web Services Security (WS-Security)
- Security Assertion Markup Language (SAML)
- WS-Trust
- WS-SecureConversation
- WS-SecurityPolicy

- WS-Policy
- WS-PolicyAttachment

As part of custom design standards (such as the standards that are established in the Service Contract Design Standards and Service Logic Design Standards precepts explained in Chapter 9), the use of SOA-related security design patterns can be regulated.

Examples of SOA design patterns that specifically address security concerns include:

- Data Confidentiality [512]
- Data Origin Authentication [515]
- Direct Authentication [518]
- Brokered Authentication [496]
- Exception Shielding [526]
- Message Screening [534]
- Trusted Subsystem [571]
- Service Perimeter Guard [564]

Finally, security control precepts can be authored to establish regulatory requirements to counter specific types of security threats. Such precepts are typically created for preventative reasons, in order to better equip services to deal with anticipated types of attacks—or reactively, after a service has been compromised as a result of an attack.

Common forms of SOA-related security attacks include:

- Buffer Overrun
- Information Leakage
- XPath Injection
- SQL Injection
- Exception Generation
- XML Parser Attack
- Malicious Intermediary (common to on-premise and cloud-based services)
- Denial of Service (common to on-premise and cloud-based services)

- Insufficient Authorization (common to on-premise and cloud-based services)
- Virtualization Attack (common to cloud-based services)
- Overlapping Trust Boundary (common to cloud-based services)

In the upcoming precept descriptions there are occasional references to security requirements and the potential involvement of the SOA Security Specialist role. However, security-specific precepts are not covered in this book due to the wide range of concerns and requirements they need to address, many of which are particular to an organization's individual needs.

SOA Governance Technology Standards

The application and enforcement of precepts documented in the upcoming chapters, as well as numerous steps within supporting processes, can be automated via SOA governance technology products. A fundamental precept governing the use of these products is that they are standardized for the regulation of services within a service inventory boundary. This means that different project teams or team members cannot use different products to perform the same governance task without prior approval.

The SOA governance technologies described in Chapter 14 are:

- Service Registries
- Repositories
- Service Agents
- Policy Systems
- Quality Assurance Tools
- SOA Management Suites

Other products mentioned include:

- Technical Editors and Graphic Tools
- Content Sharing and Publishing Tools
- Configuration Management Tools

The SOA Governance Technology Standards precept establishes general standards that are applied throughout SOA project stages to ensure consistency in the automation of governance tasks.

Metrics

A wide range of governance metrics can be collected throughout the SOA project life-cycle. This information provides important, real-world feedback for the SOA Governance Program Office and is almost always used to assess the success and effectiveness of the governance system.

Metrics of relevance to SOA governance are associated with common types of governance controls, as follows:

Cost Metrics

During initial SOA project lifecycle stages (ranging from planning to analysis), various types of cost metrics can be collected and analyzed to assess the asset value of planned investments, the cost impact of the planned adoption scope, the cost impact of encapsulating certain legacy resources with planned services, and so on.

As an example, this list contains common cost metrics used to compare the investment and operating costs of on-premise IT resources to cloud-based alternatives:

- *Up-front Costs* – The initial investment required to fund the IT resources.
- *On-going Costs* – The costs required to run and maintain the IT resources.
- *Cost of Capital* – The cost incurred as a result of raising the required funding for the IT resources.
- *Sunk Costs* – The prior investment that has been made in existing IT resources.
- *Integration Costs* – The costs required to carry out integration testing in an external (usually cloud-based) environment.
- *Locked-in Costs* – The costs associated with moving IT resources from one cloud to another.

These types of financial metrics not only help assess deployment and project scope factors, they also help stakeholders decide upon a suitable funding model for the SOA project (as covered in Chapter 4), and can even influence the choice of methodology for the overall delivery of the services.

Standards-related Precept Metrics

The most common types of metrics collected for standards-related precepts are those derived from the results of corresponding review processes.

Examples include:

- number of reviews resulting in approval
- number of reviews resulting in rejection or objection
- number of recorded standards violations
- number of requested waivers
- number of approved waivers

Collecting these statistics can help measure the effectiveness of a standards-based precept and can further help gauge whether the required standards are too strict or too flexible.

Threshold Metrics

Several of the regulations established by SOA governance precepts can put hard limitations on the design-time or runtime usage or management of a service. These types of thresholds create concrete parameters that those working with or managing services are required to respect.

Chapter 11 provides the following example thresholds associated with runtime service regulation:

- Service Composition Membership Threshold
- Service Instance Threshold
- Cloud Burst Threshold
- Service Billing Threshold
- Service Elasticity Threshold
- Service Exception Threshold
- Service Data Throughput Threshold
- Service Monitoring Footprint Threshold

Metrics derived from these types of parameters primarily relate to how well they are being adhered to.

For example:

- number of times a threshold was surpassed
- number of times the design-time usage of a service was rejected
- number of times the runtime usage of a service was rejected
- types of exceptions resulting from threshold-related service usage rejections
- number of times service usage neared a given threshold

The last type of metric in particular can be useful for planning purposes. Studying usage trends that indicate that thresholds are too close to actual required usage can help justify adjustments in the thresholds themselves (prior to services suffering from runtime performance fluctuations when thresholds are exceeded).

Vitality Metrics

Vitality metrics are collected to measure a service's on-going behavior. These metrics can, under pre-defined conditions, execute vitality triggers to help improve (or refresh) service implementations in response to new requirements.

The two types of vitality metrics discussed are:

- *Performance Metrics* – These are runtime metrics with defined thresholds that can execute various vitality triggers.
- *Compliance Metrics* – These can exist as manual or automated metrics. Manual compliance metrics are comparable to the aforementioned standards-related precept metrics. Automated compliance metrics are comparable to threshold metrics.

What distinguishes vitality metrics is that they are part of an overall vitality framework that can exist independently from other types of precepts. Vitality metrics are further explained in Chapter 13.

SUMMARY OF KEY POINTS

- There are several important precepts that are general in scope in that they can apply to multiple SOA project stages.
 - As with stage-specific precepts, the application of most general precepts results in the need for standardization.
-

CASE STUDY EXAMPLE

Raysmoore wants to ensure that all upcoming SOA investments are justified and guaranteed to advance corporate objectives. A custom general governance precept (Table 7.1) is therefore created to establish regulations for project teams to follow when soliciting and evaluating vendor proposals. This precept is not considered specific to any one SOA project stage as IT asset investments can be made during any part of a project lifecycle.

SOA Investment Proposal Precept

Objective: A financial investment in IT assets for the SOA adoption project will only be made if it is proven to advance Raysmoore's corporate strategy and business goals, as defined in the official Raysmoore Mission Statement.

<p><i>Policy:</i> Each investment proposal must specify the anticipated costs incurred and benefits generated by the proposed IT assets. In addition to mapping each proposed investment to business goals, associated costs and risks must be accurately described. Failure to comply with this policy is cause for disciplinary action.</p>	<p><i>Policy:</i> Each investment proposal must suggest at least one alternative solution that addresses the proposal requirements. The recommended solution must be justified in relation to Raysmoore's business goals and in comparison to alternatives.</p>
<p><i>Standard:</i> An SOA Investment Committee needs to be established to authorize investments above \$5,000. Investments over \$50,000 further require the approval and sign-off of the CIO.</p>	<p><i>Standard:</i> For purchases in excess of \$50,000, the recommended solution and identified alternatives must be assessed by an outside consultant.</p>
<p><i>Standard:</i> All presented investment proposals must be based on the standard Investment Proposal Template. The SOA Investment Committee is responsible for developing and maintaining this template.</p>	<p><i>Guideline:</i> When comparing solutions, corporate-approved metrics can be used along with "soft," difficult-to-measure benefits that may be relevant to associating solution feature sets to corporate business goals.</p>

Table 7.1

A custom governance precept that establishes intent and authority of a fundamental regulation.

In support of this precept, the Raysmore SOA Governance Program Office defines a set of metrics for measuring precept efficiency:

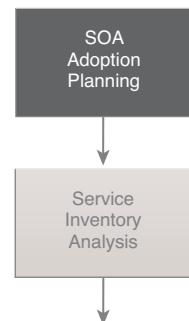
- number of proposals submitted
- time required to make a proposal decision
- number of proposals rejected due to insufficient data
- number of proposals approved
- number of proposals approved without a CIO sign-off
- size of each approved investment
- length of time required to recoup the investment
- total return generated by an approved investment at 12, 18, and 24 months
- number of investments that did not break even within 12, 18, or 24 months

The statistical data collected by these metrics will help indicate how the precept can be improved or how it may need to be better communicated.

7.3 Governing SOA Adoption Planning

Some of the most critical and repercussive decisions are made during the planning stages of an SOA adoption effort. It is during this initial stage that an approach toward adoption is defined, leading to the authoring of baseline documents, such as the SOA Adoption Roadmap and a related SOA Adoption Project Plan.

The precepts introduced in the upcoming sections establish regulations and raise considerations pertaining to the SOA Adoption Planning stage to ensure that all bases are covered when it comes to factoring in common upfront concerns. Note that the precepts themselves do not dictate the creation of SOA Adoption Roadmap and Project Plan documents; it is assumed that these fundamental specifications are being actively authored during the SOA Adoption Planning stage. Instead, the application of the upcoming precepts results in new content and intelligence that acts as critical input for these documents.



Precepts

Preferred Adoption Scope Definition

In order to assess the viability of an SOA adoption, there must be an understanding as to the initial, preferred scope of adoption. For example, project managers and stakeholders must have a preliminary idea as to whether the adoption will be enterprise-wide or limited to a domain within the enterprise. This preferred scope will influence how other precepts are defined and the extent to which related processes are carried out.

NOTE

The Preferred Adoption Scope Definition precept remains subject to change until the Service Inventory Scope precept (Chapter 8) is applied during the Service Inventory Analysis stage, at which point the actual, real-world adoption scope is determined. Both Preferred Adoption Scope Definition and Service Inventory Scope precepts are based on the Balanced Scope pillar described in Chapter 4.

Related Processes

- Organizational Governance Maturity Assessment
- Adoption Impact Analysis
- Adoption Risk Assessment

Related Roles

- Enterprise Architect
- SOA Governance Specialist

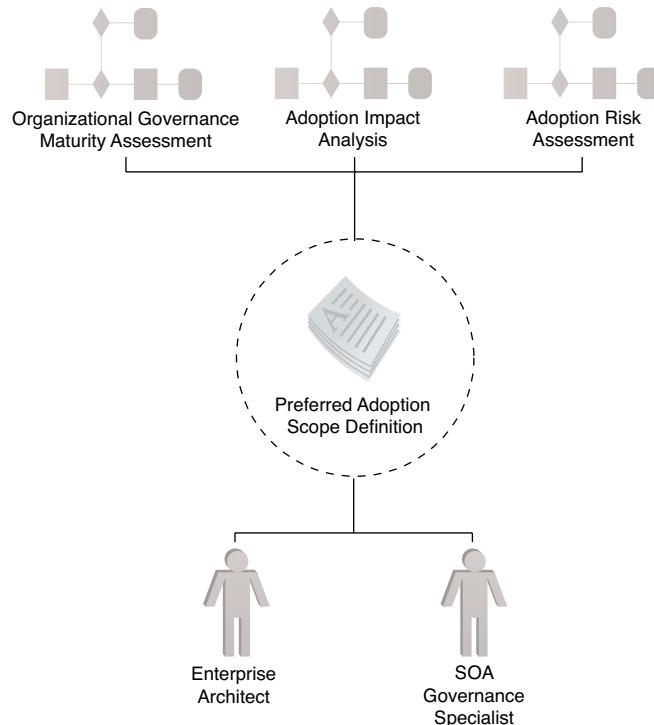


Figure 7.4

The Preferred Adoption Scope Definition precept.

Organizational Maturity Criteria Definition

A basic starting point during the planning stage is to perform a study of the IT organization, as it exists today, in order to determine its level of maturity in relation to the specific and on-going requirements of the planned SOA adoption.

In order to carry out this type of study, specific criteria needs to be defined. This criteria is based on a combination of the distinct demands and impacts associated with SOA and service-orientation, along with the specific requirements and goals of the business.

Foundational criteria is based upon three of the pillars of service-orientation introduced earlier in Chapter 4:

- Teamwork
- Education
- Discipline

These and other considerations form criteria that can be organized into a series of checklists. This precept requires that this criteria be established as an initial step in the SOA Adoption Planning stage, applicable within the scope defined via the Preferred Adoption Scope precept.

Related Processes

- Organizational Governance Maturity Assessment

Related Roles

- Enterprise Architect
- SOA Governance Specialist



Figure 7.5

The Organizational Maturity Criteria Definition precept.

Standardized Funding Model

Based on the business goals associated with the planned SOA adoption, and with further information gathered as part of the creation of the SOA roadmap, there should be enough understanding of the gap between the current state of the IT enterprise and the target state expected from the SOA initiative. In support of this, a budget needs to be established for which a corresponding funding model needs to be chosen.

Chapter 4 described the following common SOA funding models:

- Project Funding Model (Platform)
- Central Funding Model (Platform)
- Usage Based Funding Model (Platform)
- Project Funding Model (Service)
- Central Funding Model (Service)
- Hybrid Funding Model (Service)
- Usage Based Funding Model (Service)

The choice of funding model for establishing the platform (service inventory) and the delivery and evolution of individual services can have significant impacts in all areas of the initiative. Systems for methodology, management, and governance will all be influenced by that manner in which funds are allocated, the rate at which funds become available, and the source of the funds.

This precept dictates that these areas are clearly defined and established during the SOA Adoption Planning stage, because once a project is underway, unexpected changes in how funds are received and allocated can have numerous disruptive consequences.

NOTE

The chosen funding model can impact the preferred adoption scope and vice versa. Therefore, it is important to keep discussions pertaining to adoption scope inclusive of funding model definition.

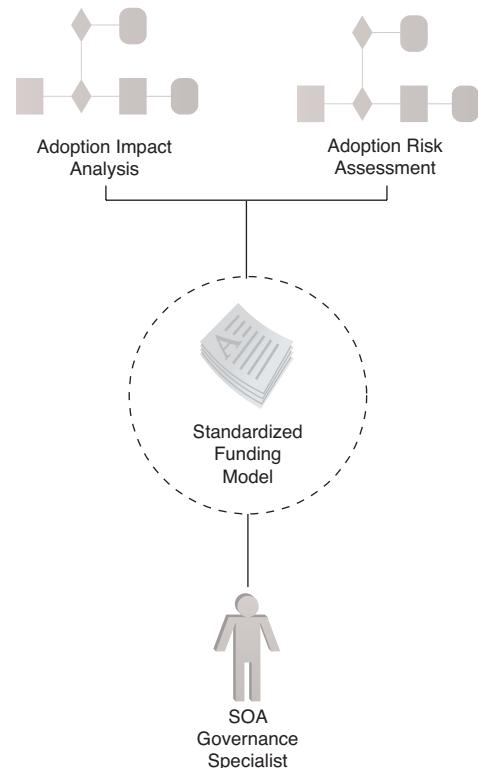


Figure 7.6

The Standardized Funding Model precept.

Related Processes

- Adoption Impact Analysis
- Adoption Risk Assessment

Related Roles

- SOA Governance Specialist

Processes

Organizational Governance Maturity Assessment

This process incorporates all of the preceding precepts in addition to several other factors with the goal of assessing the maturity of an organization in relation to different areas affected by SOA governance. The purpose of the assessment is essentially to identify how ready and prepared an organization is to assume SOA governance responsibilities. The results of completing this assessment will not only help determine strengths and weaknesses within the organization, it will also help define the appropriate scope of the SOA adoption effort to ensure that it is both meaningful and manageable.

Another expected result of this assessment process is to increase the actual level of organizational maturity.

Chapter 4 introduced the following common organizational maturity levels:

- Service Neutral
- Service Aware
- Service Capable
- Business Aligned
- Business Driven
- Service Ineffectual (negative)
- Service Aggressive (negative)

The intelligence and insight gathered by the assessment can help educate key IT stakeholders, as well as the SOA Governance Program Office itself. This can effectively transition the IT enterprise from the Service Neutral to the Service Aware maturity level, thereby reducing the risk of an SOA initiative inadvertently moving to the negative Service Ineffectual or Service Aggressive levels.

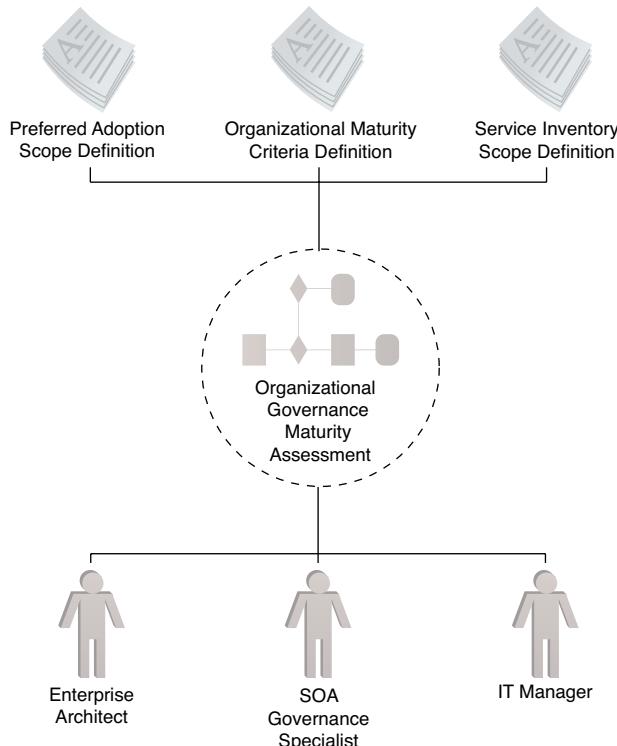


Figure 7.7

The Organizational Governance Maturity Assessment process.

NOTE

Although the Organizational Governance Maturity Assessment will be carried out by SOA Governance Specialists and others involved with the SOA Governance Program Office, the results of this assessment may end up impacting the structure of the SOA Governance Program Office itself. For example, additional resources may be added to address identified gaps or weaknesses. Or, perhaps the assessment results are lower than expected, thereby requiring a reduction in adoption scope and a corresponding reduction in the size of the SOA Governance Program Office.

To carry out an Organizational Governance Maturity Assessment process requires a framework that provides a quantitative means of measuring specific aspects of maturity in the organization. This framework may have its own structured methodology used to identify symptoms while at the same time diagnosing their root causes. Various industry and proprietary maturity assessments exist, some of which build on established IT maturity models and frameworks.

NOTE

Visit www.soaspecs.com for a list of maturity models and frameworks relevant to SOA and service-orientation. Note, however, that many of these models and frameworks are used to assess the overall maturity of an IT department or its maturity in relation to SOA adoption. Although this is helpful, it is not specific to assessing the *SOA governance* maturity of an IT department.

The Organizational Government Maturity Assessment can also be used to collect statistics about various domain-specific projects so that common pre-project maturity levels can be captured and so that common transitions between positive and negative maturity levels can be recorded. A valuable metric derived from collecting this information over time is how much easier and faster subsequent projects transition through or reach positive maturity levels compared to earlier initiatives. Measured improvements can be an indication of how early projects are raising awareness of requirements and critical success factors that can help other parts of the organization better prepare for future projects.

Related Precepts

- Preferred Adoption Scope Definition
- Organizational Maturity Criteria Definition
- Service Inventory Scope Definition (Service Inventory Analysis, Chapter 8)

Related Roles

- Enterprise Architect
- SOA Governance Specialist
- Other: IT Manager

Adoption Impact Analysis

With a solid understanding of an organization's current level of maturity in relation to the adoption of SOA and associated governance requirements, the scope of the adoption effort can be more accurately established. This will result in a clear definition of the target state that the organization expects to achieve as a result of a successful adoption.

With the current and target states well-defined, an analysis can be performed to document and assess both the technological and organizational impacts required to carry out the adoption of SOA and service-orientation.

The basic types of impacts addressed by this process are focused on:

- Cost
- Effort
- Disruption

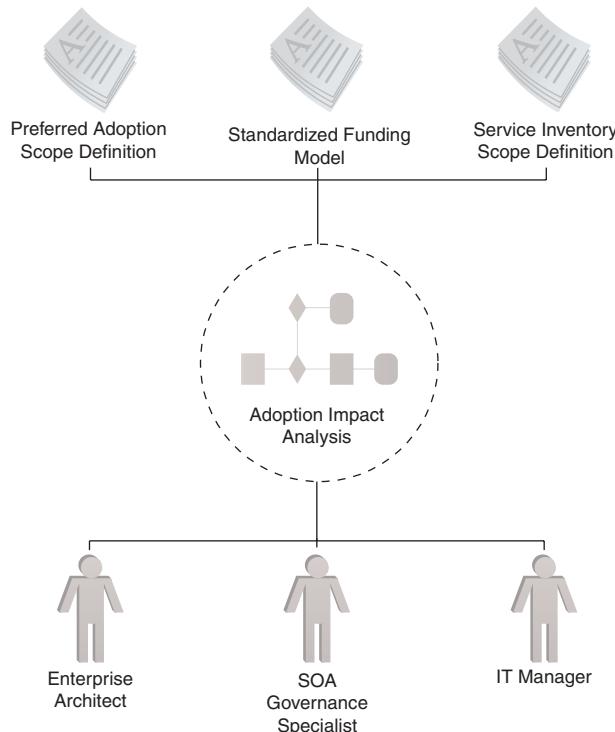


Figure 7.8

The Adoption Impact Analysis process.

The extent of this study will be directly related to the planned scope of the adoption, as well as the assessed maturity of the organization. The areas covered by an Adoption Impact Analysis process can vary dramatically, as they pertain specifically to the unique business and technology-related characteristics of a given organization's current and target state.

Provided here is a sampling of some common impact analysis areas:

- changes to traditional IT organizational and departmental structures
- introduction of new organizational roles and responsibilities
- changes to traditional IT management systems and methodologies (especially in relation to the governance, ownership, and evolution of shared services)
- impacts to legacy resources targeted for service encapsulation
- new technology products required for infrastructure and architecture upgrades
- planned shifts of on-premise IT resources to cloud environments (or vice-versa)
- new or augmented security requirements
- maturity issues related to planned service technologies
- performance and reliability impacts (especially in comparison to established silo-based applications)

The output of this process is typically a formal report detailing the cost, effort, and anticipated disruption of each identified area of impact. As with the completion of the Organizational Governance Maturity Assessment, the conclusions drawn from the completion of the Adoption Impact Analysis may lead to an adjustment in adoption scope.

NOTE

A more accurate analysis of impacts can be performed subsequent to the definition of the service inventory blueprint specification. Depending on the methodology being used for the SOA project, it may or may not be warranted to revisit and revise the results of the original Adoption Impact Analysis.

Related Precepts

- Preferred Adoption Scope Definition
- Standardized Funding Model
- Service Inventory Scope Definition (Service Inventory Analysis, Chapter 8)

Related Roles

- Enterprise Architect
- SOA Governance Specialist
- Other: IT Manager

Adoption Risk Assessment

There are several established assessment models and systems used by IT departments to help determine the risk of change. These models can be used to gauge risks associated with new technology adoption, the automation of new business processes or domains, the proposed expansion or reduction of IT staff and resources, the shifting of IT resources to external environments (such as third-party clouds), and/or the outsourcing of various IT functions.

With all of the intelligence gathered as a result of applying the preceding precepts and processes, a great amount of information will be available for stakeholders to perform a formal Adoption Risk Assessment of the planned SOA adoption project. This process may not result in any new concrete information pertaining to the project planning of the initiative; however, it may provide some fresh revelations and insights by focusing solely on risk factors.

Related Precepts

- Preferred Adoption Scope Definition
- Standardized Funding Model
- Service Inventory Scope Definition (Service Inventory Analysis, Chapter 8)

Related Roles

- Enterprise Architect
- SOA Governance Specialist
- Other: IT Manager

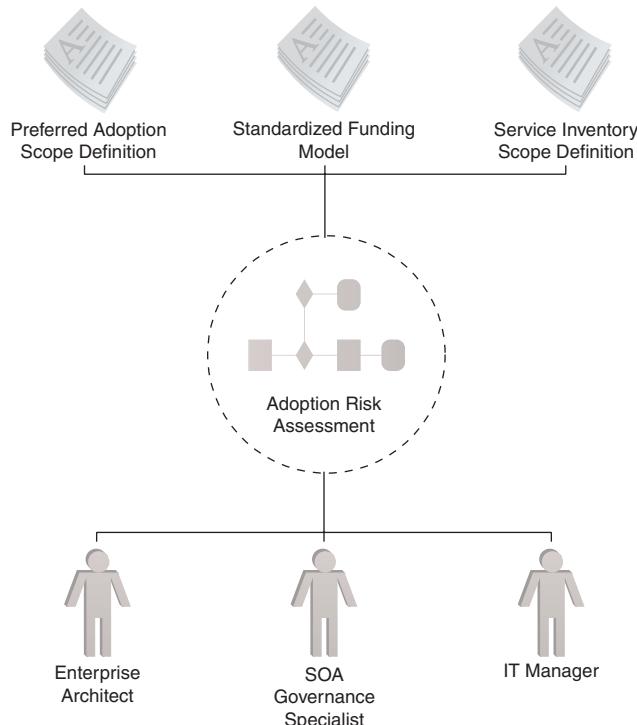


Figure 7.9

The Adoption Risk Assessment process.

People (Roles)

Enterprise Architect

Because of the broad knowledge Enterprise Architects have of the current and historical state of the IT enterprise, their involvement in most planning activities is required. Their insight into the intricacies of both organizational and technological aspects of the overall ecosystem can enable them to help define the appropriate adoption scope, assess organizational maturity, and identify impacts and risks. In many cases, they will also act as signing authorities for the SOA Roadmap and Project Planning documents.

Related Precepts

- Preferred Adoption Scope Definition
- Organizational Maturity Criteria Definition

Related Processes

- Organizational Governance Maturity Assessment
- Adoption Impact Analysis
- Adoption Risk Assessment

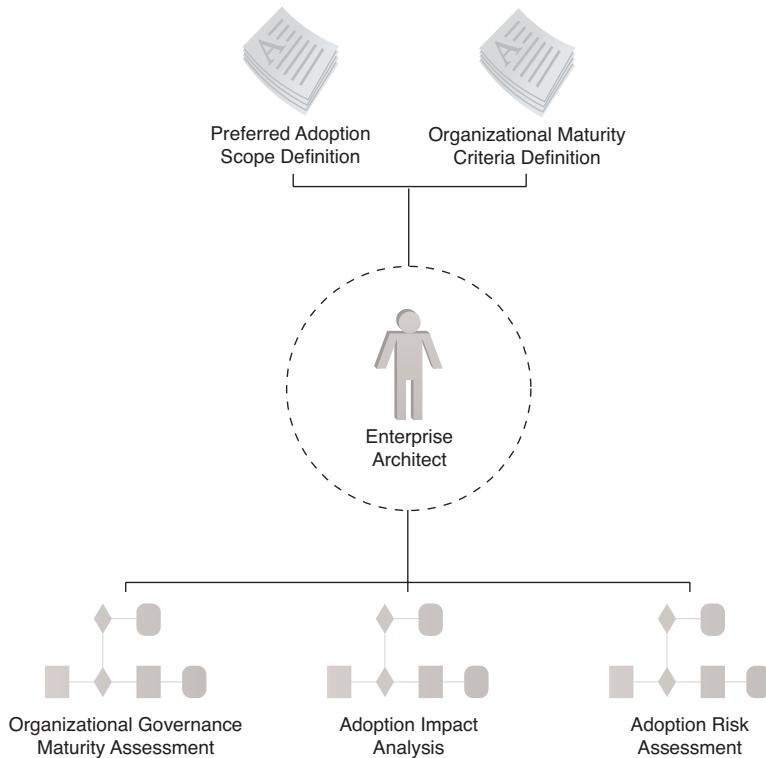
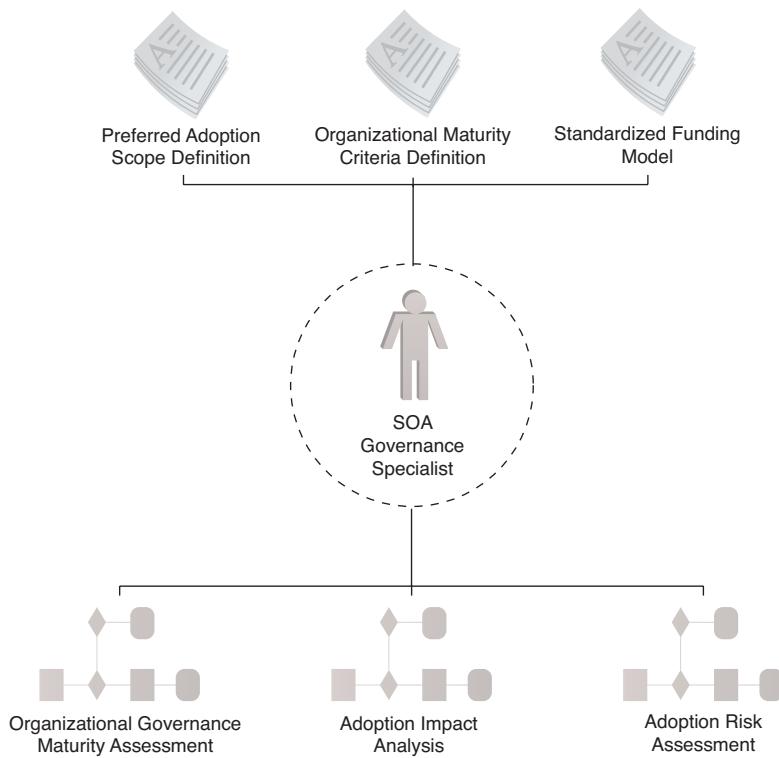


Figure 7.10

Service Adoption Planning governance precepts and processes associated with the Enterprise Architect role.

SOA Governance Specialist

On behalf of the SOA Governance Program Office (which should already have been formed prior to the SOA Adoption Planning stage), one or more SOA Governance Specialists will be actively involved in establishing precepts and processes and to further provide guidance specifically in relation to maturity considerations, impacts, and risks that pertain to pre-deployment and on-going post-deployment SOA governance requirements.

**Figure 7.11**

Service Adoption Planning governance precepts and processes associated with the SOA Governance Specialist role.

Related Precepts

- Preferred Adoption Scope Definition
- Organizational Maturity Criteria Definition
- Standardized Funding Model

Related Processes

- Organizational Governance Maturity Assessment
- Adoption Impact Analysis
- Adoption Risk Assessment

SUMMARY OF KEY POINTS

- Precepts applied during the SOA Adoption Planning stage are primarily focused on assessing the viability and suitable scope of the adoption project.
 - Several of the governance processes can result in further input that can help shape original project plans, thereby introducing iterative planning cycles.
-

CASE STUDY EXAMPLE

The IT group leading the SOA adoption project at Raysmoore has embarked on an Organizational Governance Maturity Assessment using the following COBIT-inspired rating system (Table 7.2).

Governance Maturity Score	Description
0 - Non-Existent	There is a complete lack of support and the issue has not been recognized as an area to be addressed.
1 - Initial/Ad-hoc	There is evidence that the issue has been recognized as an area to be addressed. However, there are no standardized approaches, nor are there existing approaches applied consistently.

Governance Maturity Score	Description
2 - Repeatable but Intuitive	Approaches have been developed to the stage at which common procedures are being followed by different individuals or teams undertaking the same task. However, there is no formal training or communication of standardized procedures.
3 - Defined	Procedures have been standardized, documented, and communicated. It is mandated that these approaches be followed; however, it is unlikely that deviations will be detected and corrected.
4 - Managed and Measurable	Allocated authorities monitor and measure compliance with procedures and take action when approaches appear not to be working effectively. Approaches undergo regular revision. Automation and tools may also be used.
5 - Optimized	Approaches are refined to a level of mature practice and undergo continuous revision and refinement cycles.

Table 7.2

A COBIT-based rating system.

Using this rating system, the initial phase of the SOA governance maturity assessment focuses on the following areas:

- Cultural Readiness (Table 7.3)
- Centralization Factors (Table 7.4)
- Political Environment (Table 7.5)
- Technical Project Roles and Culture (Table 7.6)

The following series of tables further document the results of individual attributes of the assessed areas. Several of the comments were collected as a result of analysis and in-person interviews with relevant representatives and stakeholders.

Cultural Readiness		
Attribute	Score	Scoring Comments
Attitude Towards Governance	3 (defined)	Raysmoore has started the process of governance centralization by having established a parent SOA Governance Program Office in support of the adoption project. There are some additional governance activities that pertain specifically to developing traditional projects, but these are not consistently used and managed across Lovelt and Reeldrill.
Attitude Towards External Solutions	4 (managed)	Raysmoore has generally demonstrated a disciplined and managed approach to reusing services from outside sources. The enterprise architecture team performed a study of available Financial solutions and selected one that is now used throughout Raysmoore's environment. On the other hand, Lovelt had to be forced to use Reeldrill's ERP service-enabled system.
Power Balance in IT	3 (defined)	Raysmoore has a defined process for working across the Lovelt and Reeldrill business domains. In some areas, such as choosing a solution, they have experience in managing this process, but most areas have not had to deal with this level of potential conflict.
Attitude Towards Long-Term Strategy	1 (ad-hoc)	The Raysmoore enterprise architecture group has been strictly limited to an advisory role and has not had any real power to make decisions and shape the organization. Its attempts at formulating a long-term strategy have been ignored by both Lovelt and Reeldrill in the past, with no consequences
Attitude Towards Innovation	1 (ad-hoc)	After assessing past projects, it is determined that openness to innovation and risk taking has not been common at Raysmoore. The group pushing innovation is the enterprise architecture team and it has often been ignored.

Table 7.3

Assessment of cultural readiness factors.

Centralization Factors		
Attribute	Score	Scoring Comments
General Centralization	3 (defined)	Raysmoore is in the process of centralizing its practices, policies, and governance activities. However, those capabilities are not yet effectively managed across its existing silos.
Geography	3 (defined)	Raysmoore is geographically distributed across various regions. Remote staff make good use of conferencing technology to maintain consistent communication.
Shared Services	3 (defined)	Shared Services already exist within Raysmoore, including services from a previous SOA project held by Reeldrill prior to its acquisition. These shared services have so far received light management.

Table 7.4

Assessment of centralization factors.

Political Environment		
Attribute	Score	Scoring Comments
Leadership	1 (ad-hoc)	Raysmoore has weak IT leadership and Lovelt leadership is resistant to change. No one group is providing leadership to initiate enterprise-wide improvements.
Empire Building	1 (ad-hoc)	There has been rampant empire building at both Lovelt and Reeldrill.
Politics	1 (ad-hoc)	There is a high degree of political influence in decision making at Raysmoore. Lovelt and Reeldrill are constantly jockeying for authority with each other.
Relationship Between Business and IT	2 (repeatable)	Some good relationships exist between the business and IT, but both would like to see improvements.

Table 7.5

Assessment of political environment factors.

Technical Project Team		
Attribute	Score	Scoring Comments
Architect Roles	1 (ad-hoc)	Architect roles are dependent upon the personality of the architect and his/her relationship with the development staff.
Architecture Acceptance	1 (ad-hoc)	Acceptance of technology architecture and design is dependent on the relationship between the architect and the development team in a given project. In some projects, technology architecture is not planned for or even referenced.
Use of Standards by Development Teams	1 (ad-hoc)	The Lovelt and Reeldrill development teams are not used to working within a larger set of enterprise standards, and do so reluctantly.
Development Team Leadership	0 (non-existent)	Development teams at both Lovelt and Reeldrill have a “hero” leadership culture. The hero programmer fixes complex problems and rejects any attempts to govern the resulting code.
Developer and Architect Roles	0 (non-existent)	There are no clearly defined roles within project teams that separate development from technology architecture. When technology architecture is made part of a specification, it is generally documented by a senior developer (or, at times, by an outsourced developer).

Table 7.6

Assessment of technical project team factors.

This preliminary glimpse at the assessment results has already raised enough red flags to prompt the SOA project leaders to initiate meetings and even training sessions with key IT managers and staff. It has become clear that further education is required to ensure that those affected by the SOA adoption project will understand the necessity of certain changes and will further be supportive of the initiative overall. It is decided that these and other identified areas of weakness must be addressed before proceeding further with any analysis stages.



Chapter 8

Governing Service Analysis Stages

8.1 Governing Service Inventory Analysis

8.2 Governing Service-Oriented Analysis (Service Modeling)

SOA PRINCIPLES & PATTERNS REFERENCED IN THIS CHAPTER

- Capability Composition [503]
- Canonical Expression [497]
- Contract Denormalization [510]
- Domain Inventory [520]
- Enterprise Inventory [522]
- Service Normalization [563]

Service analysis stages are a focal point when comparing SOA methodologies. Often, what distinguishes approaches to the delivery of services and collections of services is the extent to which analysis receives up-front attention, and how analysis is positioned in relation to other stages. From a governance perspective, the analysis of services is regulated to make the very most of the time and effort available to analysis-related project stages. The smaller the window allocated for analysis work, the greater the importance of firm governance.

In this chapter, we cover governance precepts for the two fundamental service analysis stages:

- Service Inventory Analysis
- Service-Oriented Analysis (Service Modeling)

As explained in Chapter 5, these stages are very much interrelated in that if a Service Inventory Analysis is carried out, it will establish a cycle in which the Service-Oriented Analysis stage will be performed iteratively (see Figure 8.2).

The maximum amount of iterations that can be executed is determined by the scope of the planned service inventory. The amount of *actual* iterations executed before post-analysis project stages take place is determined by the methodology. Therefore, it is important to understand the governance impact of allocating more or less time and effort to up-front analysis.

Specifically, the extent to which the Service Inventory Analysis cycle is iteratively (and competently) carried out has a direct bearing on the extent of pre-design and post-deployment governance burden services can impose.

Specifically, when more up-front time and effort is allocated:

- there is increased up-front burden, because the scope of governance responsibilities and required regulation for analysis tasks is broadened, but...
- there is a decrease in post-deployment governance burden, because the more advance effort invested in the definition of the service inventory blueprint, the better the quality of the resulting service candidates.

Better quality service candidates reduce the likelihood that subsequently delivered services will require refactoring, versioning, or will otherwise introduce avoidable governance-related impact to the IT enterprise in response to business change. In other words, increasing analysis effort helps increase the lifespan of service versions and can reduce the post-implementation governance burden of entire collections of services.

On the other hand, when going with a methodology that limits or minimizes the amount of time and effort allocated to service analysis stages:

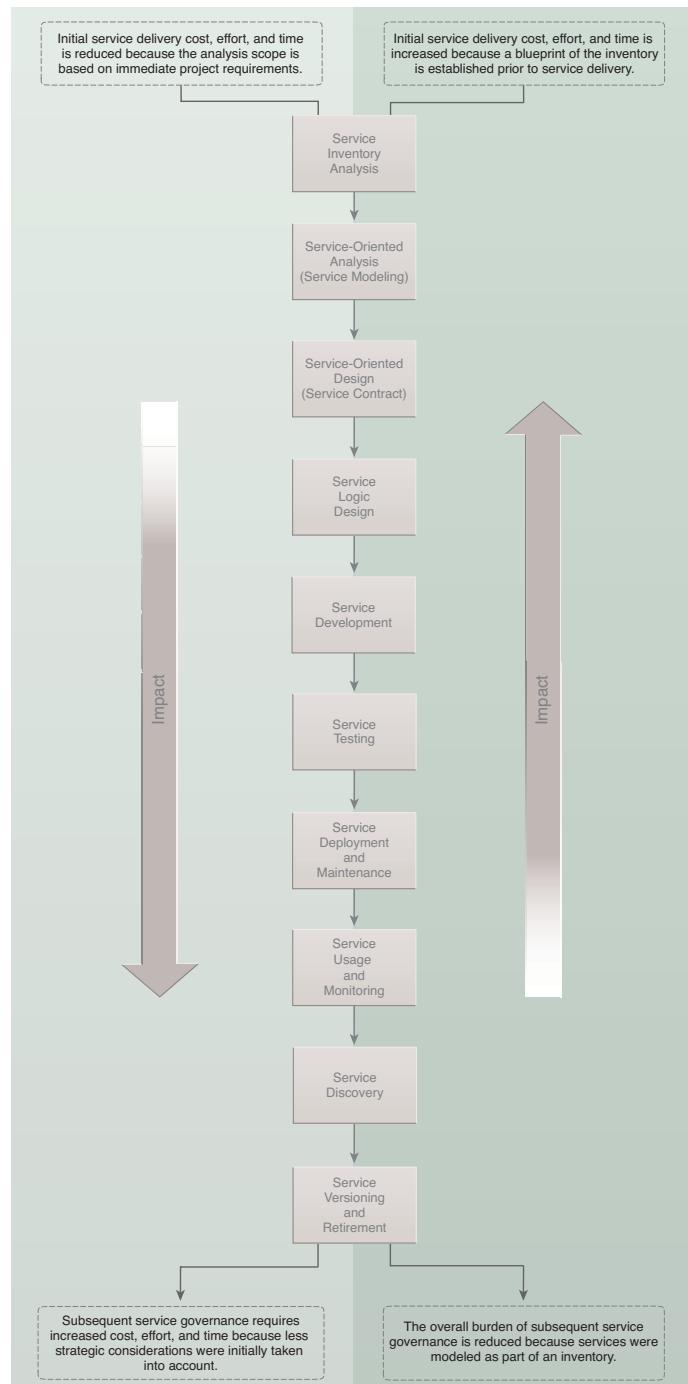
- there is decreased up-front governance burden, because there is less analysis activity to regulate, but...
- there is an increase in post-deployment governance burden because of a greater likelihood that delivered services will require versioning and refactoring sooner than later.

Figure 8.1 contrasts these approaches. The large vertical arrows represent where, within the project lifecycle stages, this type of governance impact typically occurs.

Although it is often preferred to proceed with a Service Inventory Analysis prior to a business process-specific Service-Oriented Analysis, it is not always required or possible. There are cases where practical concerns or tactical business requirements take precedence over the strategic benefits of increased up-front analysis. As stated earlier, when less time within an SOA project is allocated to service analysis stages, it further amplifies the need for strong governance of the analysis effort.

Figure 8.1

Generally, the less time and effort spent on the up-front service analysis, the greater the on-going, post-deployment governance burden. The approach on the left is comparable with bottom-up service delivery and the approach on the right is more akin to top-down delivery. SOA methodologies that attempt to combine elements of both approaches also exist.



8.1 Governing Service Inventory Analysis

As further shown in Figure 8.2, the Service Inventory Analysis is commonly organized into iterative cycles whereby the Service-Oriented Analysis process is repeatedly completed (as originally explained in Chapter 5).

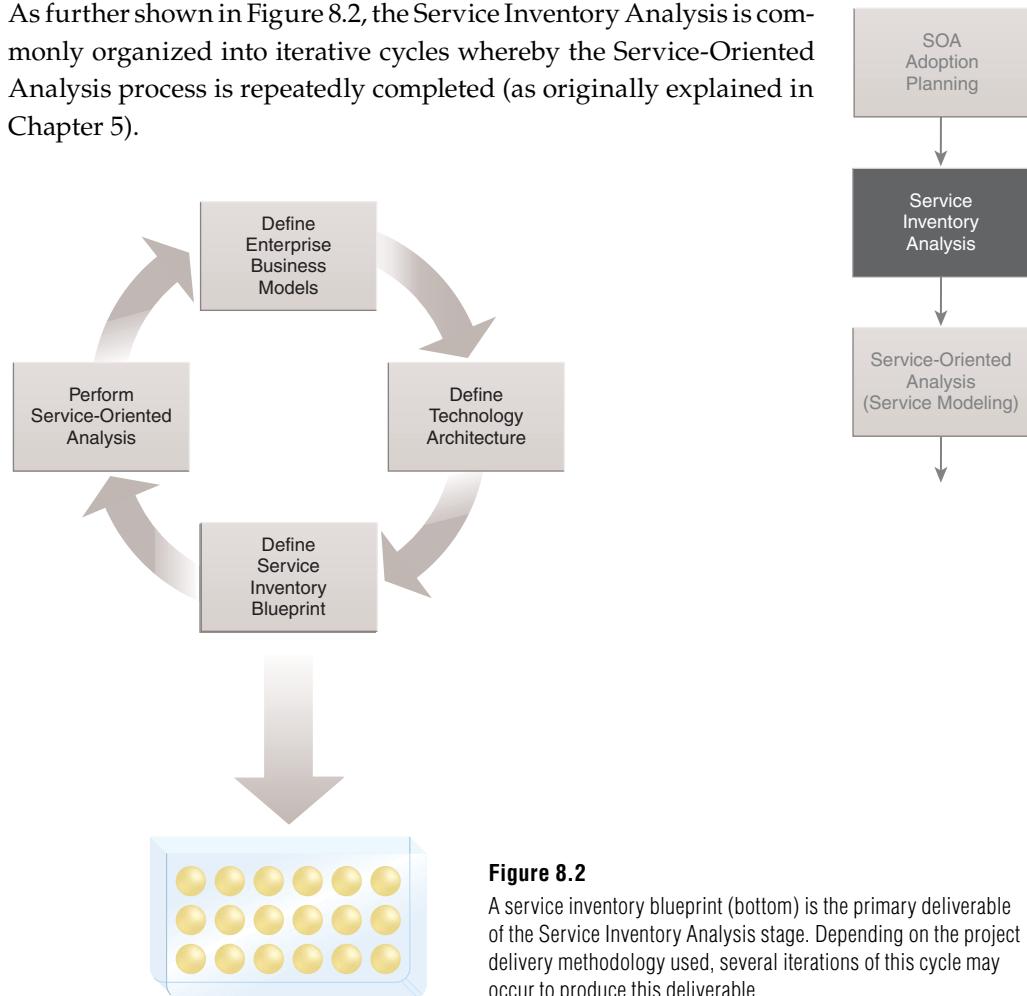


Figure 8.2

A service inventory blueprint (bottom) is the primary deliverable of the Service Inventory Analysis stage. Depending on the project delivery methodology used, several iterations of this cycle may occur to produce this deliverable.

NOTE

The *Define Enterprise Business Models* step in the Service Inventory Analysis lifecycle refers to published business specifications, documents, and artifacts that provide suitable input for the Service-Oriented Analysis stage. Several of the precepts covered in Chapter 12 pertain to the definition of these types of business models as the basis of governance precepts.

Precepts

Service Inventory Scope Definition

A key success factor in any Service Inventory Analysis effort is the correct scope definition for the service inventory blueprint. As stated in the SOA Manifesto (and the *Balanced Scope* section in Chapter 4), the scope of a service inventory needs to be “meaningful and manageable.” The manageability of the planned service inventory is a foremost governance concern.

The application of this precept will need to involve the SOA Governance Program Office as this scope represents not only the magnitude of the SOA project delivery effort, but also the corresponding SOA governance effort. The definition of the service inventory scope is an overarching precept that establishes a concrete boundary within the IT enterprise for the delivery of a specific collection of services. As such, it further influences (but does not set) the scope of the Service Inventory Analysis stage.

Within the established boundary, services need to adhere to consistent governance, management, and methodology. Specifically, from a governance perspective, the service delivery lifecycle is required to adhere to the precepts established by the governance system applied to the service inventory.

Further, the service inventory blueprint scope represents a contract between IT and any related business operating units. This agreement establishes priorities that provide a means of carrying out a methodology for the identification, definition, development, and deployment of services within the service inventory in order to:

- match the value and urgency of relevant business needs
- most effectively support high-priority business goals (such as improving operational efficiency, profitability, organizational agility, etc.)
- cost-effectively leverage associated IT skills and resources

To achieve a governance system dedicated to Service Inventory Analysis with the aforementioned qualities depends on the successful usage and adherence of a number of additional stage-specific governance processes and precepts.

SOA PRINCIPLES & PATTERNS

This precept relates to the Enterprise Inventory [522] and Domain Inventory [520] patterns, each of which provides an alternative approach for the definition of a service inventory scope, in relation to the scope of the overall IT enterprise.

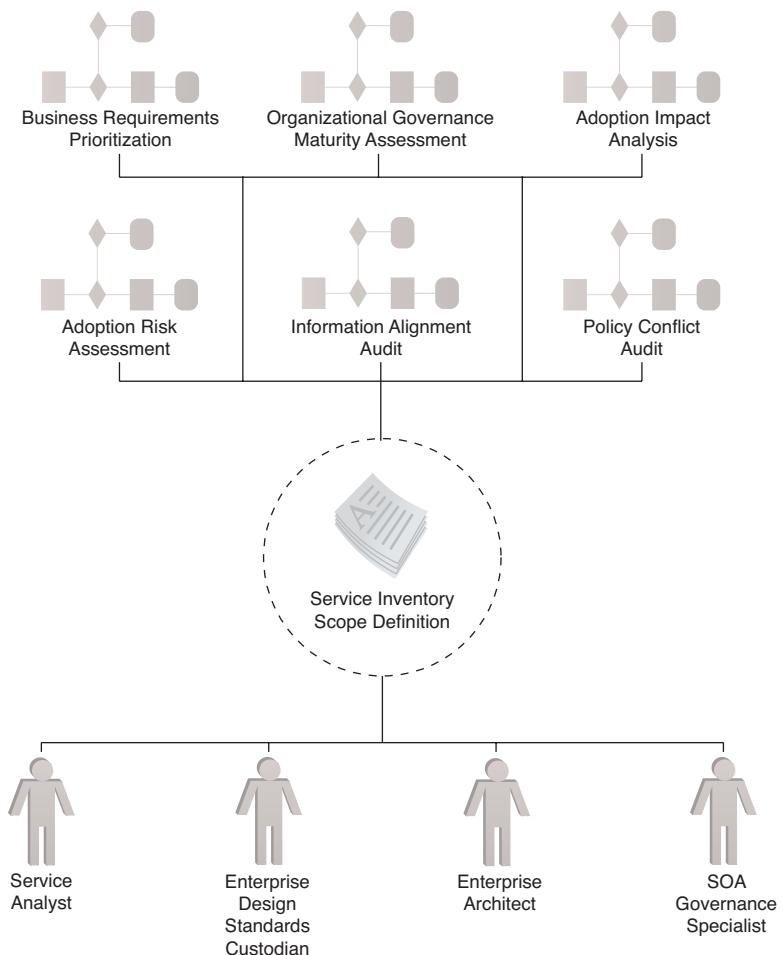


Figure 8.3

The Service Inventory Scope Definition precept.

Related Processes

- Business Requirements Prioritization
- Organizational Governance Maturity Assessment (Chapter 7)
- Adoption Impact Analysis (Chapter 7)
- Adoption Risk Assessment (Chapter 7)

- Information Alignment Audit (Chapter 12)
- Policy Conflict Audit (Chapter 12)

Related Roles

- Service Analyst
- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Governance Specialist

Processes

Business Requirements Prioritization

A primary consideration when carrying out the Service Inventory Analysis is how to prioritize business processes and requirements in order to determine the order in which they are subjected to the iterations of the Service-Oriented Analysis process. This relates directly to the *Define Enterprise Business Models* step that starts off each iteration of the Service Inventory Analysis cycle.

Business requirements prioritization involves a process whereby the business automation requirements within the domains or sub-domains of the Service Inventory Analysis scope are compared. This comparison can involve various criteria, including the urgency of requirements, cost of automation, impact on legacy systems, and so on.

A common deliverable used to organize the typical output of this process is the *business heat map*. This document contains tables consisting of columns that represent individual business units, each further containing a set of business activities (or responsibilities) specific to that unit. Heat maps are organized into business domains. For most large organizations, many of these domains will be split into sub-domains, each of which may be controlled by a separate IT Manager. In most cases, the top one or two domain levels suffice.

In order to fulfill its responsibilities, each domain has to be able to carry out certain business activities. Assigning a business activity to a specific domain implies assigning ownership of that activity to that domain. However, it does not imply that it is exclusively involved in the execution of that activity because many business activities can involve interaction between multiple business domains. Developing the heat map begins by creating a list of the business activities owned by each domain.

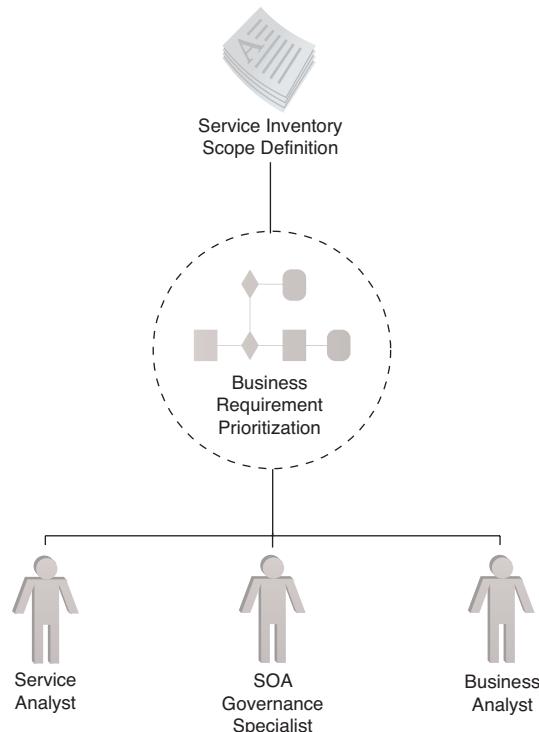


Figure 8.4
The Business Requirement Prioritization process.

The next step is to define which of those identified business activities needs to be improved or enhanced in some way to meet specific strategic business goals.

Examples include:

- business activities that are strategic but are being performed ineffectively
- business activities that reflect new business opportunities
- business activities that address new threats
- business functionality performance improvements
- areas of business activity that need to be downsized
- business activities that might be candidates for outsourcing or selling off

“Hot spot” areas, such as these, that require specific attention (based on the aforementioned criteria) are displayed in a different color. Separate explanatory text describes the type of attention of each focus area. (See the case study at the end of this section for an example of a business heat map.)

A business heat map can provide valuable governance input to help plan and carry out a Service Inventory Analysis. It represents a consensus view of business priorities and offers the additional benefit of providing a high-level view of the internal structure of one or more business domains.

Related Precepts

- Service Inventory Scope Definition

Related Roles

- Service Analyst
- SOA Governance Specialist
- Other: Business Analyst

People (Roles)

Service Analyst

The Service Inventory Analysis stage is essentially led by Service Analysts. Depending on how many are involved with the creation of a given service inventory blueprint, it may be necessary to designate a lead Service Analyst. This may especially be required when multiple teams of Service Analysts are working concurrently on producing service candidates for the same service inventory blueprint.

On the other hand, Service Analysts tend to take a secondary role when involved with the Business Requirements Prioritization process. Their contribution to this process primarily relates to ensuring a constant alignment between business processes and business domains used as input for the Service Inventory Analysis and Service-Oriented Analysis stages.

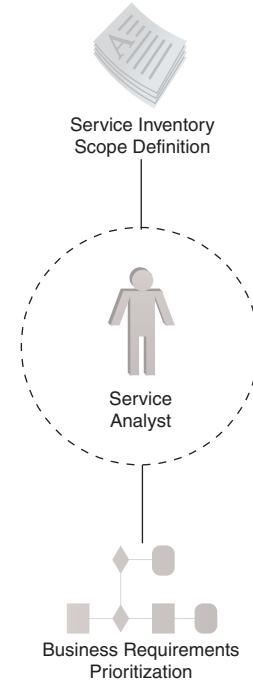


Figure 8.5

Service Inventory Analysis governance precepts and processes associated with the Service Analyst role.

Related Precepts

- Service Inventory Scope Definition

Related Processes

- Business Requirements Prioritization

Enterprise Design Standards Custodian

Certain types of design standards can impose limitations or parameters that affect the definition of service inventory boundaries. For example, there may be product-specific platforms or proprietary technology or legacy resources that can impose hard perimeters that end up reducing or adjusting the planned scope of a service inventory. Enterprise Design Standards Custodians will be aware of these limitations (and may have themselves even defined them), making their involvement in this stage important. As a role that also commonly audits compliance to enterprise design standards, their participation may, in fact, be mandatory in that they may need to sign off on the proposed service inventory blueprint specification.

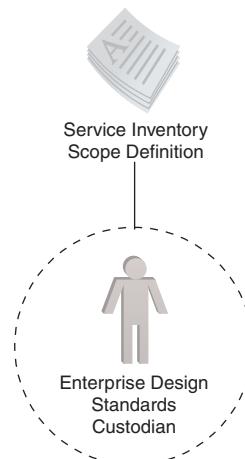


Figure 8.6

Service Inventory Analysis
governance precepts and processes
associated with the Enterprise
Design Standards Custodian role.

Related Precepts

- Service Inventory Scope Definition

Related Processes

N/A

Enterprise Architect

Whereas the Enterprise Design Standards Custodian role is primarily concerned with the impact of and compliance to custom design standards, the Enterprise Architect will have a broader understanding of the overall IT ecosystem affected by the planned service inventory architecture. This insight will be helpful for providing guidance from a practical perspective, especially in relation to legacy resources that may need to be encapsulated by services within the planned service inventory scope.

Related Precepts

- Service Inventory Scope Definition

Related Processes

N/A

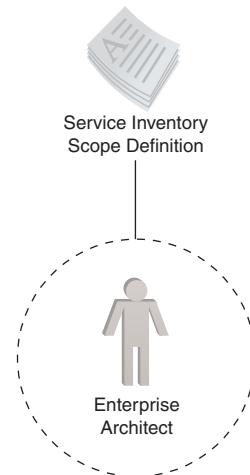


Figure 8.7

Service Inventory Analysis
governance precepts and
processes associated with the
Enterprise Architect role.

SOA Governance Specialist

The primary responsibility of the SOA Governance Specialist during the Service Inventory Analysis stage is to provide input as to the governance requirements and impacts that the proposed service inventory scope (and associated business requirements scope) will introduce. Furthermore, this role will be assigned the task of ensuring that the Service Inventory Scope Definition precept and Business Requirements Prioritization process are carried out according to other pre-defined compliance criteria.

Related Precepts

- Service Inventory Scope Definition

Related Processes

- Business Requirements Prioritization

SUMMARY OF KEY POINTS

- The definition of the service inventory scope correspondingly determines the scope of the Service Inventory Analysis effort and further establishes a concrete boundary in which a collection of services will subsequently be delivered.
 - Business requirements prioritization helps determine the order and sequence of business processes and requirements that are processed through iterations of the Service Inventory Analysis cycle, which includes iterations of the Service-Oriented Analysis process.
-



Figure 8.8
Service Inventory Analysis
governance precepts and
processes associated with the
SOA Governance Specialist role.

CASE STUDY EXAMPLE

Subsequent to a series of executive board meetings, Raysmoore issues a new mission statement that identifies the following as the primary strategic business goals for the upcoming year:

- reduce business operating costs to increase overall profitability
- complete the creation of a seamless supply chain across all Raysmoore subsidiaries
- improve the ability of Raysmoore to monitor and control its supply chain
- improve the speed with which Raysmoore and its subsidiaries can respond to legislative changes
- improve Raysmoore's ability to respond to business opportunities through acquisitions, business partnerships, and outsourcing

The SOA governance program created by the SOA Governance Program Office for the first planned service inventory includes the Service Inventory Scope precept, as shown in Table 8.1.

Service Inventory Scope Precept	
<i>Objective:</i> Define a balanced service inventory scope.	
<i>Policy:</i> Ensure that the scope is meaningful.	<i>Policy:</i> Ensure that the scope is manageable.
<i>Standard:</i> Require the scope to be meaningfully cross-silo by encapsulating at least five well-defined business processes.	<i>Standard:</i> Require the scope to be realistically manageable by calculating cost, determining the length of required up-front analysis effort, and getting written approval from all affected IT Managers.
	<i>Guideline:</i> Use the Business Requirements Prioritization process to help define and refine the service inventory scope and to further assist with determining required up-front analysis effort.

Table 8.1

The Service Inventory Scope precept, as defined by Raysmoore's governance office.

Business Analysts within Raysmoore's strategic planning group intend to follow this precept by starting with the recommended Business Requirements Prioritization process. A team of Business Analysts and Service Analysts is assembled to carry out a prioritization of the corporation's business requirements across the Raysmoore, Lovelt and Reeldrill business domains.

The results of this effort are the heat maps displayed in Figures 8.9, 8.10, and 8.11. While there appears to be duplication across the heat maps, it is acknowledged that Raysmoore controls and coordinates the supply chain of its multiple subsidiaries, each of which is responsible for the scope of its own operations.

The initial plan was to establish one central enterprise service inventory that encompasses all of the business domains identified in each of the three business heat maps. However, when the group attempts to get sign-off from all of the affected IT Managers (as required by one of the precept standards), many objections arise. It becomes evident that for the scope of the service inventory to become manageable, it must be reduced.

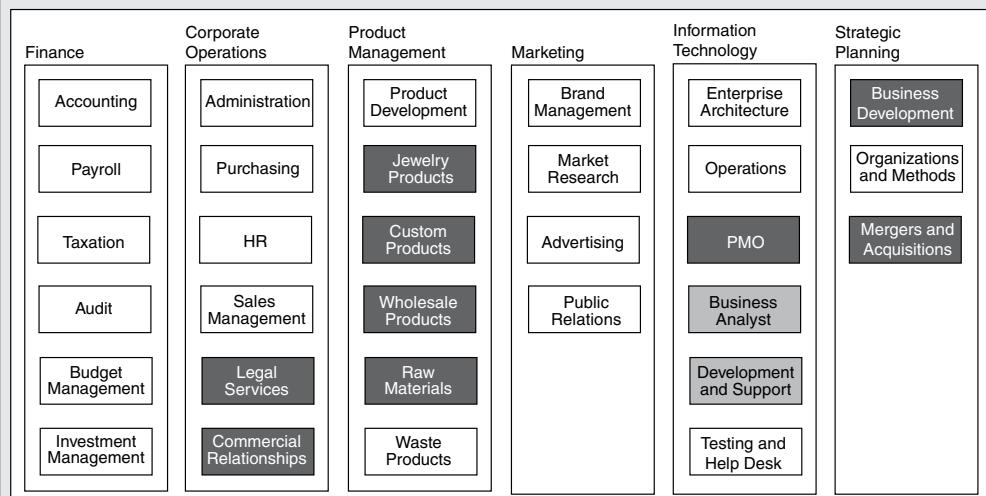
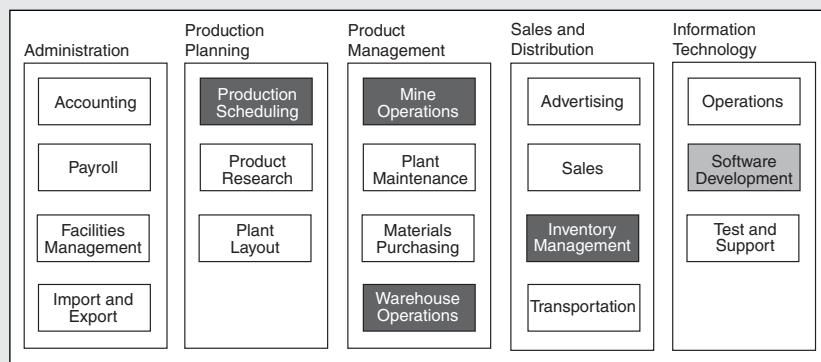
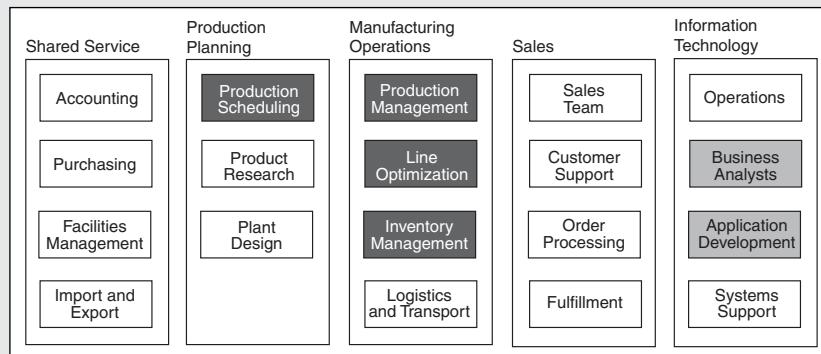


Figure 8.9

The Raysmoore business heat map.

**Figure 8.10**

The Lovelt business heat map.

**Figure 8.11**

The Reeldrill business heat map.

The group is determined to establish a level of federation across the Raysmore IT enterprise and those of its two subsidiaries (Lovelt and Reeldrill). After further analysis based on the hot spots identified in the heat maps, the service inventory scope is limited to production and operations areas, which encompass the following business units:

- Raysmore Product Management
- Lovelt Production Management
- Reeldrill Manufacturing Operations

The result is the definition of the Productions and Operations domain service inventory.

After sign-off from IT Managers and other required stakeholders, the group turns its attention back to the business heat maps where the following business activities are identified as the immediate priority for the Service Inventory Analysis:

- Raysmore Jewelry Products (retail)
- Raysmore Custom Products (retail)
- Raysmore Wholesale Products
- Raysmore Raw Materials
- Lovelt Mine Operations
- Lovelt Warehouse Operations
- Reeldrill Production Management
- Reeldrill Line Optimization
- Reeldrill Inventory Management

These business activities reflect the perceived need to streamline development support activities and reduce duplication of effort as a means of achieving the first strategic goal of reducing operating costs. The Raysmore team feels that the creation of federated, cross-silo services will help integrate Lovelt and Reeldrill's supply chains. To meet all of the strategic business goals is likely going to require enhancements to both the processes and tools involved in the business activities.

Specifically, in order to meet the third strategic business goal, the Raysmore product management groups are required to improve their ability to monitor all aspects of the product supply chains across subsidiaries. This activity needs to be closely aligned with the enhancements to the subsidiaries' production management supply chain processes.

Because of the risk of heavy fines for non-compliance with increasingly complex government legislation, the Raysmore Business Development and Legal Services teams are given the task to create an improved compliance process.

Finally, Raysmore Business Development, Mergers and Acquisitions, Commercial Relationships and PMO groups are given the mission of creating a more structured and reliable process for integrating future acquisitions, outsourcing, and enhanced commercial relationships.

The board recognizes that these are ambitious goals and that there are several dependencies between them. Because the constraints on investment are tight, the strategic planning team has been given the authority to approve or reject all new proposed development projects with budgets exceeding \$100,000 on the basis of the degree to which they support these business priorities. The first of these projects to receive approval is for the optimized automation of the cross-subsidiary supply chain.

8.2 Governing Service-Oriented Analysis (Service Modeling)

The following precepts and processes pertain specifically to the Service-Oriented Analysis project stage, regardless of whether Service-Oriented Analysis is carried out as part of the Service Inventory Analysis cycle.

Precepts

Service and Capability Candidate Naming Standards

Of the various service modeling conventions that may exist, having a system for the consistent labeling of service candidates and service capability candidates is important for governance purposes.

The naming established when individual service candidates are defined will need to be compatible with service candidates defined by other project teams during separate iterations of the Service-Oriented Analysis process. Further, service and capability names will carry over to the Service-Oriented Design process where these names then become solidified as part of the service's physical design.

Related Processes

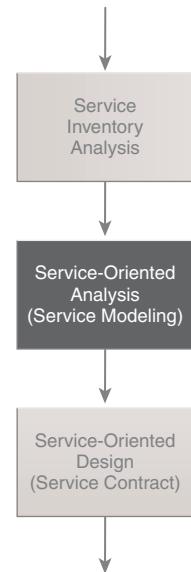
- Service Candidate Review

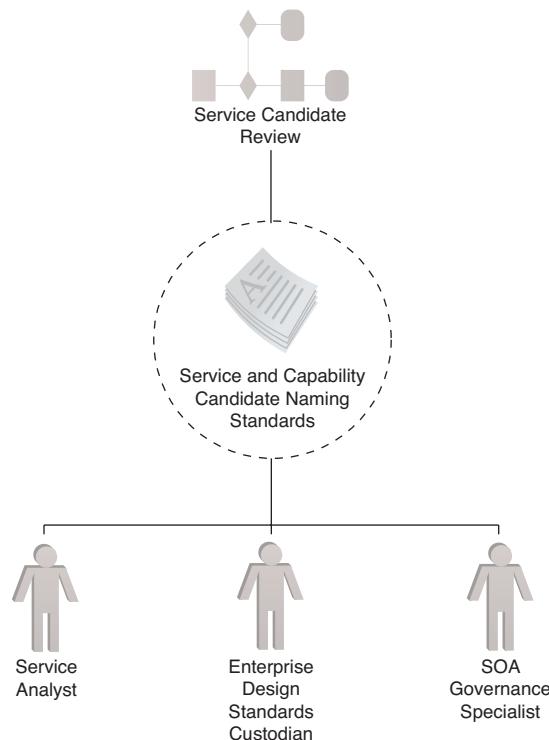
Related Roles

- Service Analyst
- Enterprise Design Standards Custodian
- SOA Governance Specialist

SOA PRINCIPLES & PATTERNS

This precept relates to Canonical Expression [497], a pattern commonly applied to establish service naming standards.



**Figure 8.12**

The Service and Capability Candidate Naming Standards precept.

Service Normalization

The Service Normalization precept dictates that service candidates within a given service inventory cannot have overlapping boundaries. This guarantees that no two services will introduce redundant logic, which maximizes reuse opportunities for shared services and forces services to compose other services when functionality outside of their boundaries is required.

Although the need for this precept typically emerges during the Service-Oriented Analysis stage when individual business process

SOA PRINCIPLES & PATTERNS

This precept relates to the Service Normalization [563] and Capability Composition [503] patterns, both of which are concerned with establishing independent functional contexts for services and preserving these functional boundaries within the scope of a service inventory.

definitions are being decomposed, it is a precept that actually affects and applies to the service inventory blueprint as a whole. Therefore, this precept can also be associated with the Service Inventory Analysis stage.

Related Processes

- Service Candidate Review

Related Roles

- Service Analyst
 - Service Architect

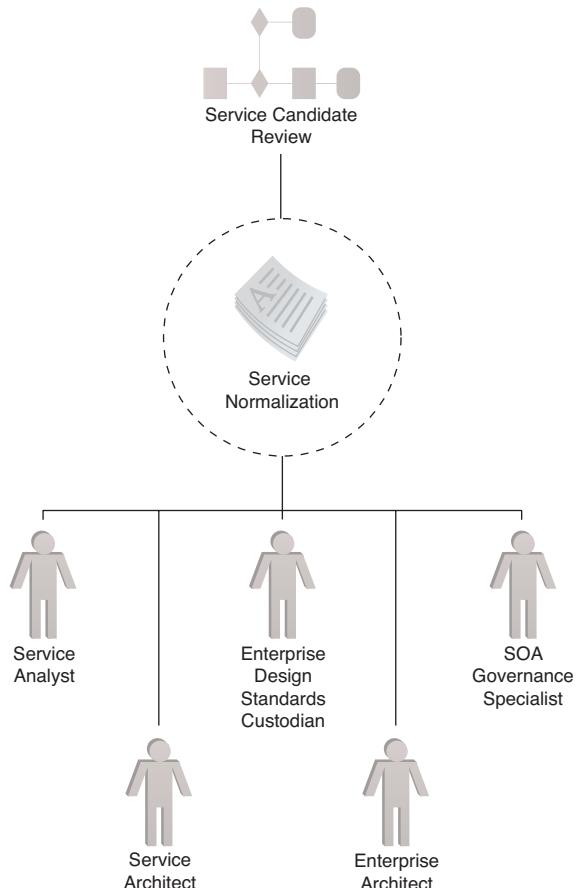


Figure 8.13
The Service Normalization precept.

- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Governance Specialist

Service Candidate Versioning Standards

Though services are only conceptualized during the Service-Oriented Analysis stage, the need for versioning can still arise when service candidates and even entire service inventory blueprints are required to be re-aligned with how deployed services have been changed or versioned. There may even be a service candidate versioning system that facilitates the versioning of service candidates during the Service Inventory Analysis cycles. In this case, agnostic service candidates undergo repeated refinement as a result of being reviewed as part of multiple service composition candidate definitions.

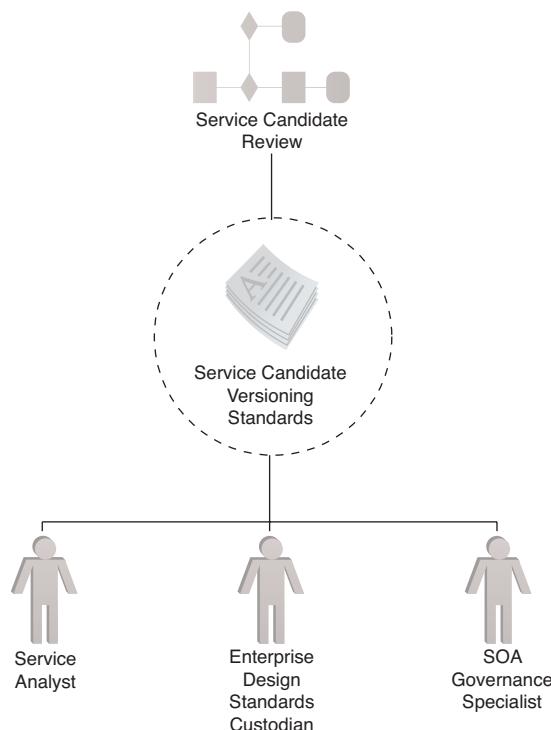


Figure 8.14

The Service Candidate Versioning Standards precept.

Related Processes

- Service Candidate Review

Related Roles

- Service Analyst
- Enterprise Design Standards Custodian
- SOA Governance Specialist

Processes

Service Candidate Review

A formal review is recommended when a service candidate is first defined and whenever it undergoes significant changes.

Possible outcomes of the service candidate review are:

- changes or extensions to the service candidate are approved
- approval of the changes or extensions is deferred pending additional remedial service modeling activity
- changes to the service are rejected

In some cases, reviews may be required for individual service capability candidates. This requirement may surface when service capabilities have different custodians or when an already implemented service is extended by the addition of one or more new service capabilities that are first modeled prior to actual design and development. In the latter case, the review may also check for compliance to service candidate versioning standards.

Related Precepts

- Service and Capability Candidate Naming Standards
- Service Normalization
- Service Candidate Versioning Standards

Related Roles

- Service Analyst
- Service Architect
- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Governance Specialist

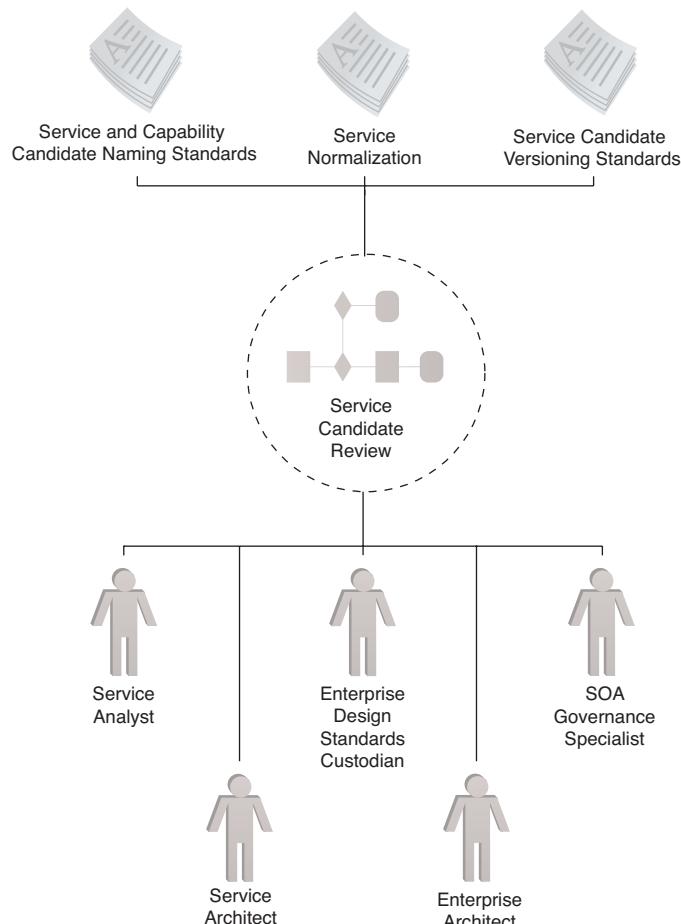


Figure 8.15

The Service Candidate Review process.

People (Roles)

Service Analyst

Service Analysts are typically involved with both the application and definition of precepts and processes that pertain to the Service-Oriented Analysis stage. Because of their hands-on participation in service modeling processes, they have the highest level of expertise required to help establish modeling standards in cooperation with SOA Governance Specialists and Enterprise Design Standards Custodians. Although it will generally be a Service Analyst that proposes one or more modeled services for the Service Candidate Review, it is common for a peer Service Analyst to participate as a reviewer as well.

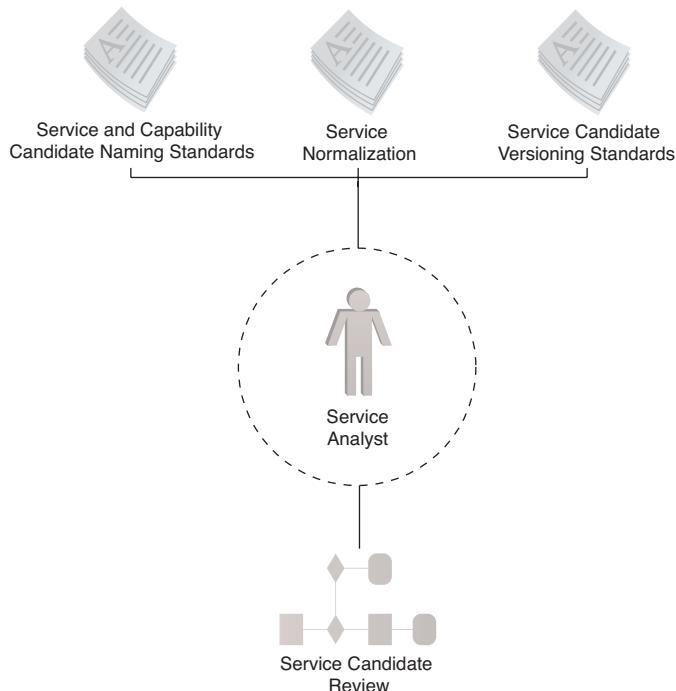


Figure 8.16

Service-Oriented Analysis governance precepts and processes associated with the Service Analyst role.

Related Precepts

- Service and Capability Candidate Naming Standards
- Service Normalization
- Service Candidate Versioning Standards

Related Processes

- Service Candidate Review

Service Architect

The involvement of Service Architects with the Service Normalization precept is typically more peripheral than Service Analysts. Service Architects can assist with the definition and application of these precepts by providing input regarding the practical considerations of establishing functional service boundaries. This can influence the extent to which some service candidates can be normalized as well as the granularity of their functional boundaries. The same practical issues can require a Service Architect to act as one of the reviewers during the Service Candidate Review process.

Related Precepts

- Service Normalization

Related Processes

- Service Candidate Review

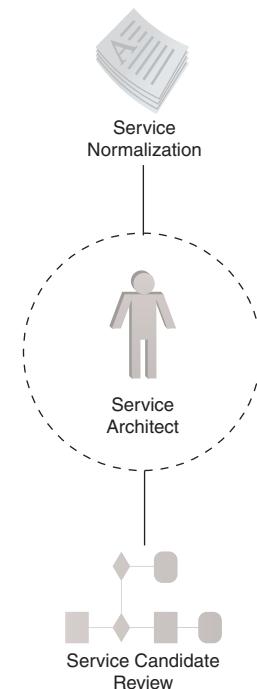


Figure 8.17

Service-Oriented Analysis governance precepts and processes associated with the Service Architect role.

Enterprise Design Standards Custodian

The definition of any standards pertaining to service modeling and Service-Oriented Analysis in general will require the involvement or, at minimum, the approval of the Enterprise Design Standards Custodian. To verify compliance to these standards during the Service Candidate Review may further require attendance by the person assuming this role; however, it is not uncommon for the Enterprise Design Standards Custodian to delegate this responsibility to a senior Service Analyst.

Related Precepts

- Service and Capability Candidate Naming Standards
- Service Normalization
- Service Candidate Versioning Standards

Related Processes

- Service Candidate Review

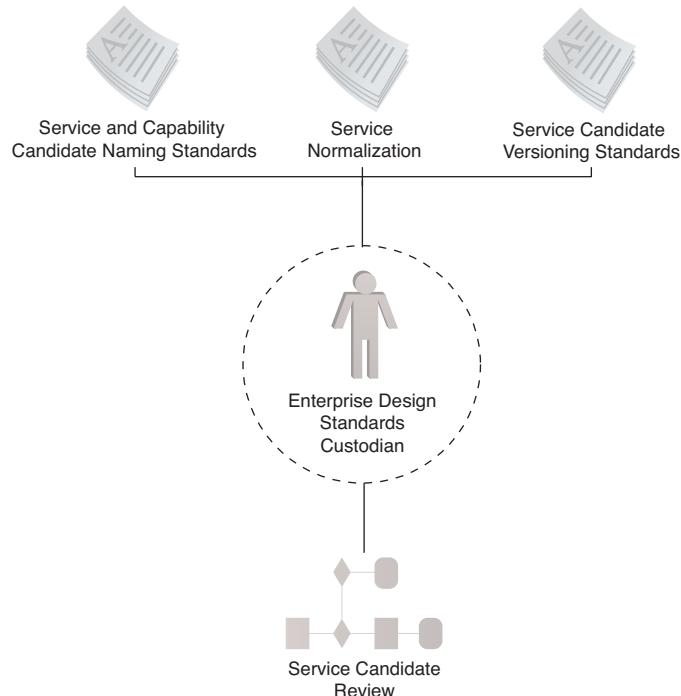


Figure 8.18

Service-Oriented Analysis governance precepts and processes associated with the Enterprise Design Standards Custodian role.

Enterprise Architect

Whereas Service Architects can provide input regarding service-specific encapsulation considerations, Enterprise Architects can comment on broader platform and resource issues that can further affect the application and definition of the Service Normalization precept. For the same reasons, an Enterprise Architect may need to serve on the review team for the Service Candidate Review process.

Related Precepts

- Service Normalization

Related Processes

- Service Candidate Review

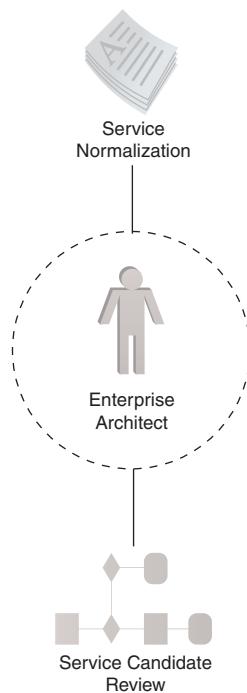


Figure 8.19

Service-Oriented Analysis governance precepts and processes associated with the Enterprise Architect role.

SOA Governance Specialist

Bringing together the precepts and a supporting process for the Service-Oriented Analysis stage falls upon the SOA Governance Specialist. A primary task in accomplishing this is coordinating the involvement of Service Analysts, Service Architects, and possibly also an Enterprise Architect and Enterprise Design Standards Custodian.

Especially challenging can be the incorporation of the precepts with a methodology that only allows limited or partial service analysis, prior to proceeding with post-analysis project stages. In this case, judgment is required to ensure that the most important standards are adhered to and that some form of meaningful review can occur before service candidates are transposed to concrete service contract designs.

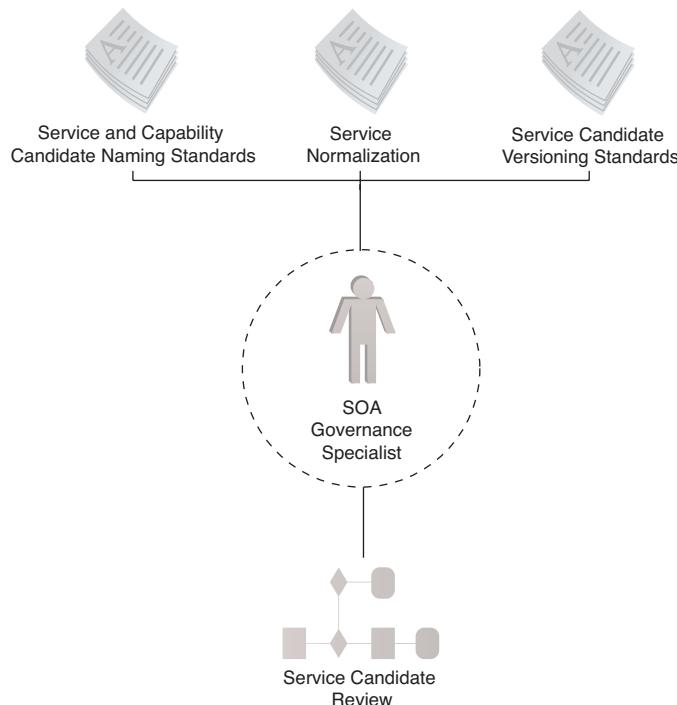


Figure 8.20

Service-Oriented Analysis governance precepts and processes associated with the SOA Governance Specialist role.

Related Precepts

- Service and Capability Candidate Naming Standards
- Service Normalization
- Service Candidate Versioning Standards

Related Processes

- Service Candidate Review

SUMMARY OF KEY POINTS

- The Service-Oriented Analysis stage is responsible for producing the very first incarnations of services and service capabilities, and therefore presents an opportunity to establish precepts that can support eventual governance tasks.
 - The primary governance responsibilities relate to the consistent definition and versioning of service candidates, and to ensuring their review before moving on to the Service-Oriented Design stage.
-

CASE STUDY EXAMPLE

The first iteration of the Service-Oriented Analysis process focuses on the decomposition of the business process that encompasses supply chain management across Raysmore, Lovelt, and Reeldrill. Several service candidates and service capability candidates are derived from the subsequent service modeling effort. The Product service is an agnostic service candidate that in particular appears to have high reuse potential throughout the planned service inventory.

Initially, Business Analysts encountered confusion caused by different subsidiary legacy environments using different terms to refer to the same types of functionality and, worse, the same terms referring to different types of functionality.

Fortunately, the SOA governance program includes the Service and Capability Naming precept (Table 8.2) that addresses this problem. The SOA Governance Program Office, in cooperation with Service Analysts, supplement this precept by authoring a lexicon that encompasses a vocabulary of naming conventions for service candidates and service capability candidates.

Service and Capability Naming Precept	
<p><i>Objective:</i> Service candidates and service capability candidates within the same service inventory must adhere to the same naming conventions.</p>	
<p><i>Policy:</i> Ensure that a common vocabulary is used.</p>	<p><i>Policy:</i> Ensure that a common name format is used.</p>
<p><i>Standard:</i> Require that all service candidates and service capability candidates are assigned names based on a pre-defined lexicon and/or pre-defined naming conventions.</p>	<p><i>Standard:</i> Require that pre-defined formats are applied to all service and service capability candidate names so that the name structure and sequence of combined words is consistent.</p>
<p><i>Standard:</i> Require that all service and service capability names are reviewed for compliance in accordance with the vocabulary and format standards as part of the Service Candidate Review process.</p>	

Table 8.2

The Raysmore Service and Capability Naming precept.

The required usage of this precept further ensures that service modeling tasks carried out by different project teams remain in alignment. Newly defined service candidates will be able to encapsulate and abstract disparate legacy applications (and the disparity among the terms and vocabulary used by these legacy environments), and still provide a conceptual set of services that establish standardized endpoints.

This level of consistency among service candidates allows Raysmore to consolidate similarities across supply chain business requirements from its subsidiaries and essentially enables the definition of service candidates that represent parts of supply chain processes as logical wholes. However, as service modeling efforts proceed, Business Analysts begin to uncover some significant differences between the requirements of Raysmore and Lovelt in relation to supply chain business rules used for authorization and access to product inventory data. Specifically, Lovelt has traditionally allowed customers to view its product inventory levels, whereas Raysmore has always had a policy that regarded overall corporate stock levels as a trade secret available only to internal staff.

This conflict affects the definition of the Product service that is being modeled. Initially, the project team proposes to create two variations of the service, one specific to Raysmore and the other specific to Lovelt (Figure 8.21). This would allow each service to incorporate different business rules that could be further independently evolved.

**Figure 8.21**

Two service candidates with overlapping functional boundaries.

SOA Governance Specialists, however, push back on this approach and state that ultimately this service inventory is intended to establish a unified view of business automation that spans the Raysmoore and Lovelt IT enterprises. Further, they point to the Service Normalization precept (Table 8.3), which states that the creation of separate service candidates with comparable functional boundaries is not allowed.

Service Normalization Precept	
<p><i>Objective:</i> Service candidates within the same service inventory cannot have overlapping functional boundaries.</p>	
<p><i>Policy:</i> Ensure that no two services within the same service inventory contain redundant logic.</p>	<p><i>Policy:</i> Ensure that no two services within the same service inventory have overlapping functional contexts (regardless of their actual implemented logic).</p>
<p><i>Standard:</i> Require that the logic of all service candidates is explicitly documented in service profiles.</p>	<p><i>Guideline:</i> Register service candidates in the service registry with a status of “analysis.”</p>
<p><i>Standard:</i> Require that the functional context of each service candidate is reviewed in relation to other services in the service inventory blueprint, as part of the Service Candidate Review process.</p>	

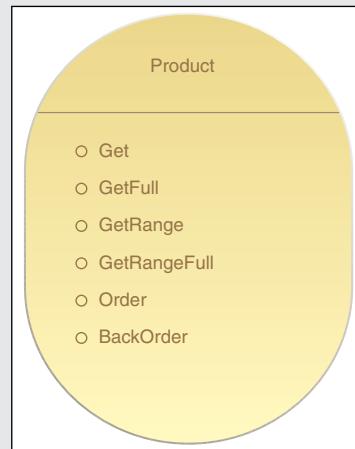
Table 8.3

The Raysmoore Service Normalization precept.

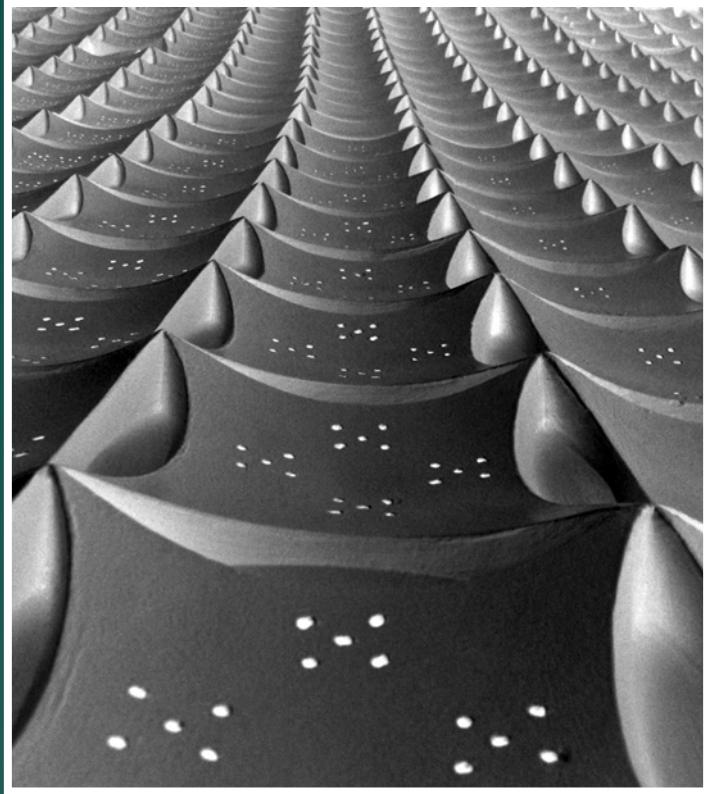
Subsequent to further analysis and discussion, it is decided to only proceed with a single Product service (Figure 8.22). The conflict is addressed by the application of the Contract Denormalization [510] pattern, which allows for a single service to contain capabilities that express redundant logic without being in violation of the Service Normalization precept. Access control of the individual service capabilities is further ensured via appropriate security mechanisms.

Figure 8.22

A single Product service candidate containing service capability candidates that provide an extent of redundant functionality. The capabilities further qualified with "Full" allow for the retrieval of product inventory stock values for Lovelt customers only.



Chapter 9



Governing Service Design and Development Stages

- 9.1 Governing Service-Oriented Design (Service Contract)**
- 9.2 Governing Service Logic Design**
- 9.3 Governing Service Development**

SOA PRINCIPLES & PATTERNS REFERENCED IN THIS CHAPTER

- Canonical Expression [497]
- Canonical Protocol [498]
- Canonical Schema [500]
- Concurrent Contracts [508]
- Contract Centralization [509]
- Decoupled Contract [517]
- Dual Protocols [521]
- Legacy Wrapper [532]
- Metadata Centralization [536]
- Schema Centralization [551]
- Service Abstraction (478)
- Service Autonomy (481)
- Service Composability (486)
- Service Discoverability (484)
- Service Façade [558]
- Service Loose Coupling (477)
- Service Reusability (479)
- Service Statelessness (482)
- Standardized Service Contract (475)
- Validation Abstraction [574]

All of the project stages so far lead up to the actual concrete design and creation of the service architecture and service logic. The stages in this chapter result in the realization of services as IT assets, which makes their regulation a critical success factor.

Governance considerations addressed by the upcoming precepts focus primarily on establishing and validating compliance to standards. While various organizational roles are involved with specifics pertaining to precepts and processes in particular stages, governance professionals participate in all aspects of service design and development regulation to ensure continuity and consistency from prior analysis stages through to subsequent deployment and usage stages.

9.1 Governing Service-Oriented Design (Service Contract)

Service contracts within a given service inventory boundary are intended to establish a federated service endpoint layer within which services can intrinsically interoperate (as per the Increased Federation and Increased Intrinsic Interoperability goals described in Chapter 3). Therefore, having governance controls to help keep service contracts consistent and in alignment with each other is crucial to the success of an SOA initiative.

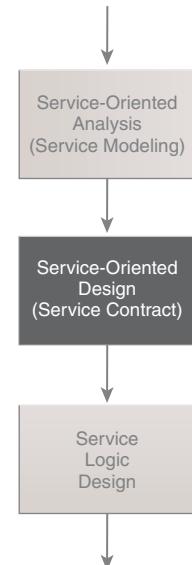
Precepts

Schema Design Standards

This precept is dedicated to standardizing the use of common schemas by services and applications, inter-schema sharing and linking, as well as constructs and elements used within schemas.

Standards that relate to the design of common schemas and data models used by service contracts can be part of the overall Service Contract Design Standards precept. A separate Schema Design Standards precept is required when shared schemas are part of an independent data architecture that may be used by other parts of the IT enterprise, outside of service boundaries.

For example, a common design standard is that schemas defining data models for common business documents be canonical across a pre-defined scope or domain. An Invoice schema used by an Invoice service may also be used by a middleware broker managing the exchange of invoice data between disparate legacy accounting applications.



Note that this precept may also address the design of single-purpose or service-specific schemas. In this case, specific standards are created to govern the use of schema syntax, functions, validation logic, and structure. Additionally, these standards may further indicate when service-specific schemas must still make use of common schemas—and—for REST service design, Schema Design Standards may also address uniform contract aspects, such as the custom standardization of media types (see the upcoming *Schema Custodian* section for examples).

SOA PRINCIPLES & PATTERNS

This precept is directly related to the Canonical Schema [500] and Schema Centralization [551] patterns. Note that the aforementioned Invoice schema example demonstrates an application of both the Canonical Schema [500] and Canonical Data Model [Hohpe, Woolf] patterns.

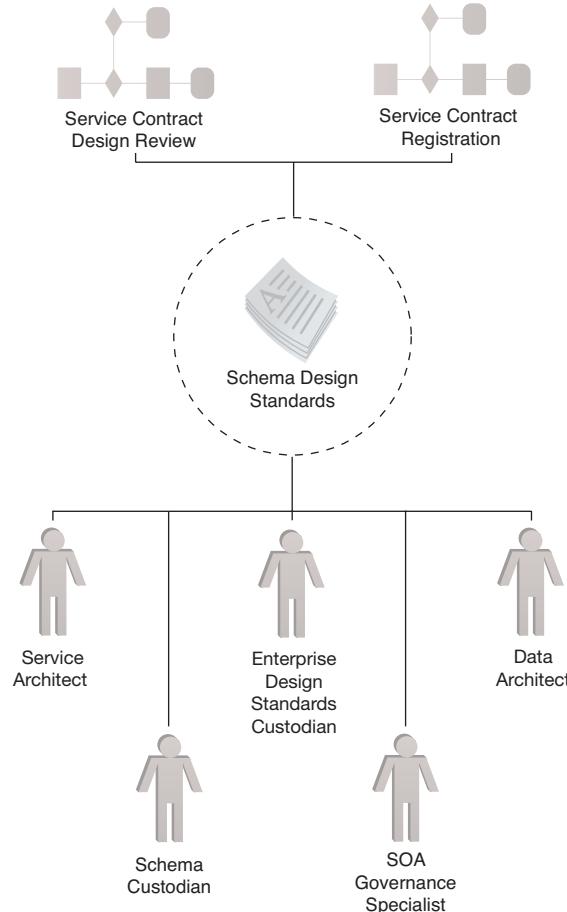


Figure 9.1

The Schema Design Standards precept.

Related Processes

- Service Contract Design Review
- Service Contract Registration

Related Roles

- Service Architect
- Schema Custodian
- Enterprise Design Standards Custodian
- SOA Governance Specialist
- Other: Data Architect

NOTE

Corresponding precepts that govern policy design further establish common standards. Precepts and processes that pertain to both business and operational policy design are covered in Chapter 12.

Service Contract Design Standards

Custom design standards need to be in place to ensure that the technical contracts for services within a given service inventory adhere to established and required conventions.

Common examples of custom service contract design standards include those that:

- identify allowed service contract-related industry standards and languages (such as WSDL, WADL, SOAP, etc.)
- identify what parts of allowed industry standards and languages can and cannot be used
- require compliance to custom or industry determined use of industry standards and languages (such as WS-I profiles)

SOA PRINCIPLES & PATTERNS

This precept can relate to any patterns that shape or standardize service contract characteristics. Common examples include the Decoupled Contract [517], Contract Centralization [509], Concurrent Contracts [508], Canonical Protocol [498], Dual Protocols [521], Canonical Expression [497], and Validation Abstraction [574] patterns.

Furthermore, concerns addressed by custom Service Contract Design Standards are often closely tied to the application of the Standardized Service Contract (475), Service Loose Coupling (477), Service Abstraction (478), and Service Discoverability (484) design principles, as explained in the upcoming *Service-Orientation Architecture Design Standards* section.

- position and/or limit the use of canonical data models and schemas within service contracts
- position and/or limit the use or quantity of service-specific schemas
- position and/or limit the use of shared or service-specific policies
- establish naming and syntax conventions
- address communications quality requirements in support of service interpretability and discoverability
- custom design standards that address the method usage and resource identifier syntax for uniform contracts used by REST services
- custom design standards that address the usage of media types for REST services

This precept may optionally introduce architectural standards that relate to how the service contract (as a physical artifact) is positioned within the overall service technology architecture. These types of standards do not define the underlying technology architecture; they only establish certain requirements that may impact how the service contract is designed independently.

NOTE

Depending on how precepts are applied and associated with corresponding review processes, it may be more practical to broaden the scope of the Service Contract Design Standards precept to encompass any of the following related precepts:

- Schema Design Standards
- Service Metadata Standards (Chapter 12)
- Business Policy Standards (Chapter 12)
- Operational Policy Standards (Chapter 12)
- Service-Orientation Contract Design Standards
- SLA Template

Related Processes

- Service Contract Design Review
- Service Contract Registration
- Policy Conflict Audit (Chapter 12)

Related Roles

- Service Architect
- Schema Custodian
- Policy Custodian
- Enterprise Design Standards Custodian
- SOA Security Specialist
- SOA Governance Specialist

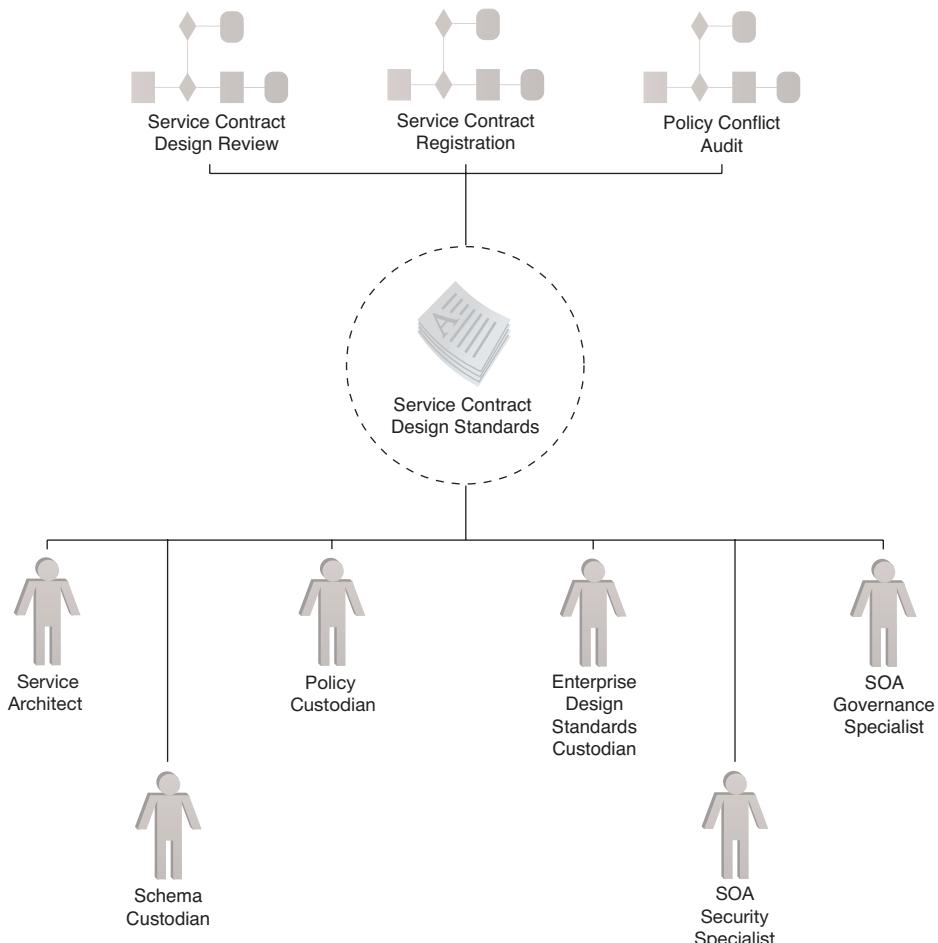


Figure 9.2

The Service Contract Design Standards precept.

Service-Orientation Contract Design Standards

The service-orientation paradigm establishes eight design principles that, when applied, shape software programs into units of service-oriented logic so that they are equipped with specific characteristics that help realize the strategic goals of service-oriented computing (see the *Service-Orientation* and *Service-Oriented Computing* sections in Chapter 3). A subset of these design principles specifically affects the design and positioning of the service contract, and are therefore highly relevant to the governance of the Service-Oriented Design project stage.

In addition to the aforementioned custom design standards, a meaningful level of compliance needs to be ensured with the following contract-related service-orientation design principles:

- Standardized Service Contract (475)
- Service Loose Coupling (477)
- Service Abstraction (478)

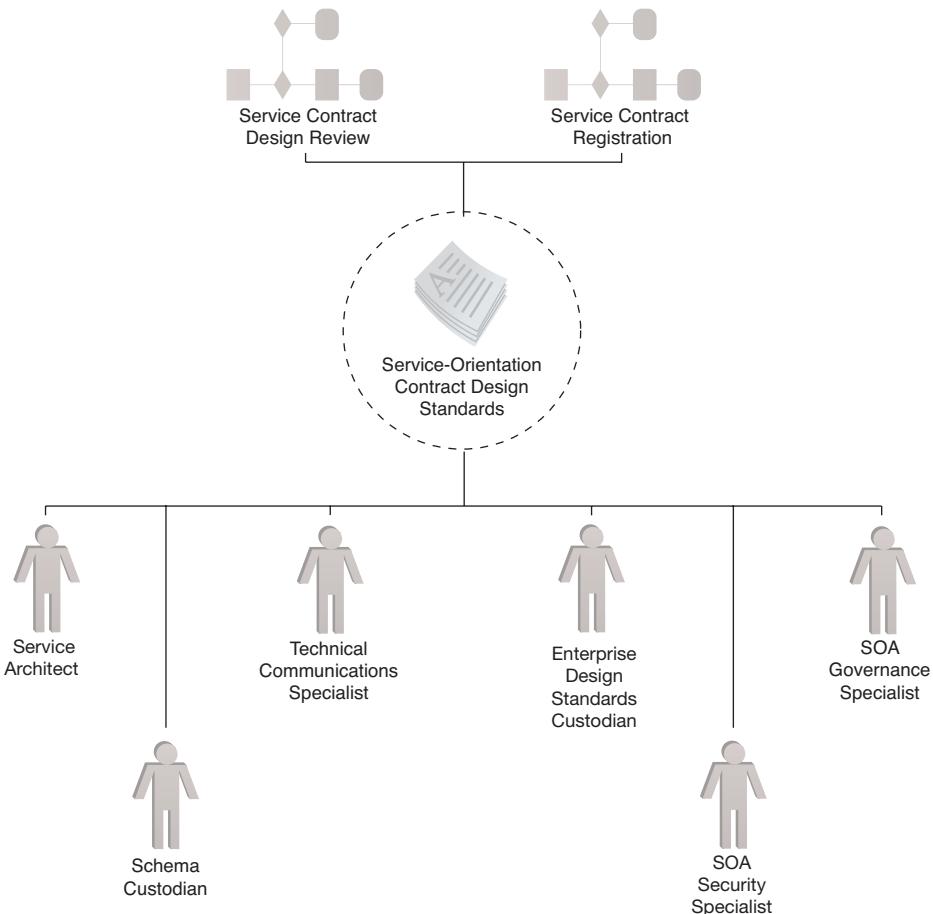
As mentioned shortly, when associated with the *Service Contract Registration* process, compliance to the Service Discoverability (484) principle is also measured during this stage.

Related Processes

- Service Contract Design Review
- Service Contract Registration

Related Roles

- Service Architect
- Schema Custodian
- Technical Communications Specialist
- Enterprise Design Standards Custodian
- SOA Security Specialist
- SOA Governance Specialist

**Figure 9.3**

The Service-Orientation Contract Design Standards precept.

SLA Template

The SOA Governance Program Office needs to mandate the use of a standard template for human-readable service-level agreements (SLAs). The template can include required and optional parts, the former of which is commonly dedicated to expressing availability and reliability guarantees.

A different SLA template may be needed for cloud-deployed services in order to conform to proprietary conventions of the third-party cloud provider, or to address service-level guarantees that are distinct to the cloud environment. Every effort should be made

to add unique parts to the SLA template as extensions to the same base template used for all services.

Related Processes

- Service Contract Design Review
- Service Contract Registration

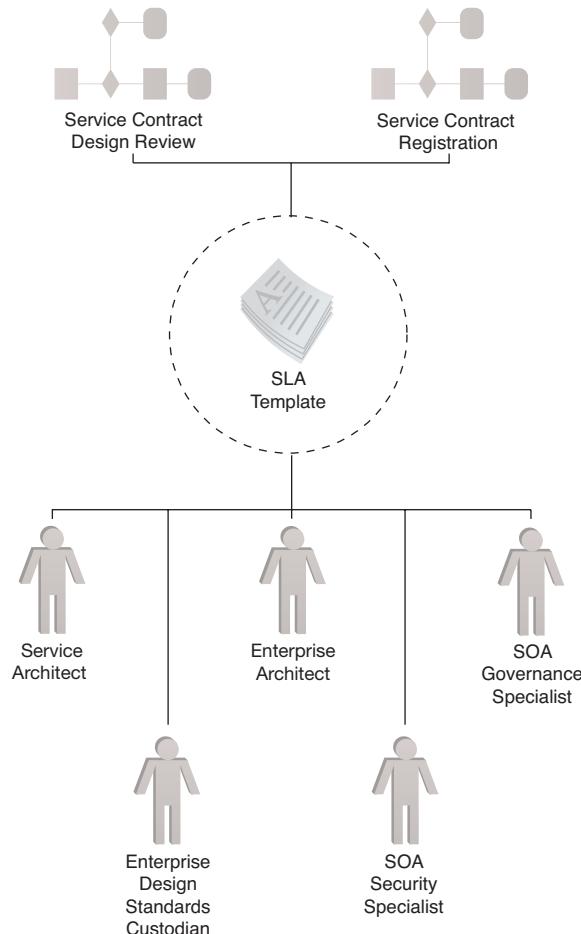


Figure 9.4
The SLA Template precept.

Related Roles

- Service Architect
- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Security Specialist
- SOA Governance Specialist

Processes

Service Contract Design Review

A service contract design review is the principal process for governing the quality and completeness of the service contract, as well as its required compliance to design standards. The SOA Governance Program Office can establish a checklist to be used during this review to ensure consistency so that the service logic (that will be subsequently constructed during the Service Development stage) will conform to the specified contract when it is delivered.

The purpose of the review is to confirm that the service contract meets design standards, and truly reflects the functional scope originally defined during analysis stages. Each aspect of the service contract must be approved by technically qualified reviewers before it is deemed ready for release. Since expensive resources may be committed to developing both the service itself and other software programs that will consume the service, based solely on the service contract, it is worthwhile to give this review significant attention.

Probable outcomes of this review are:

- The service contract is accepted and then handed over to the Service Logic Design stage, where the underlying logic and architecture of the service implementation will be specified in support of realizing the functionality expressed in the service contract.
- The service contract fails one or more of the technical or content reviews and is returned for rework.

It can be helpful to conduct an informal requirements review immediately before the Service Contract Design Review to ensure that both the functional and non-functional requirements are stable before the service contract is published. There may also be follow-up issues that need to be addressed before the service contract is actually finalized, which may result in the need for additional “mini-reviews.”

NOTE

It can be beneficial to have regular “review days” that handle all or most outstanding reviews, rather than attempting to schedule a larger number of small reviews. This approach is especially helpful with Service Contract Design Reviews, as it provides the opportunity to review several service contracts in relation to each other.

Related Precepts

- Schema Design Standards
- Service Contract Design Standards
- Service-Orientation Contract Design Standards
- SLA Template
- Service Metadata Standards (Chapter 12)
- Business Policy Standards (Chapter 12)
- Operational Policy Standards (Chapter 12)

Related Roles

- Service Architect
- Schema Custodian
- Technical Communications Specialist
- Enterprise Design Standards Custodian
- SOA Security Specialist
- Enterprise Architect
- SOA Governance Specialist

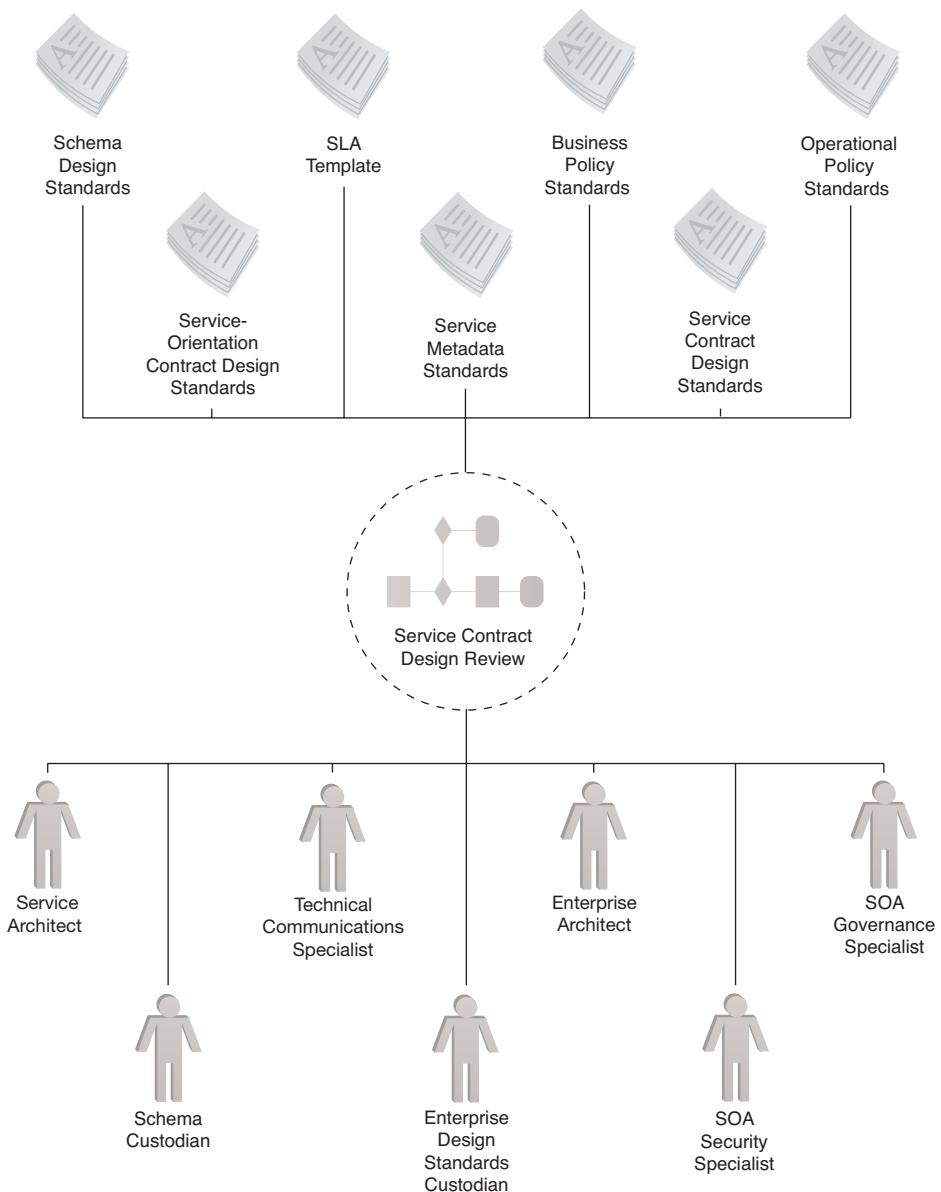


Figure 9.5
The Service Contract Design Review process.

Service Contract Registration

If the review of the service design was successful, it may be desirable, at that point, to initiate a separate process for the formal registration of service metadata in the service registry (or repository). This will give other project teams concrete information about a service that is nearing development and deployment stages. The registry information can further include a status value and a target availability date.

If a service contract is available, any of its technical or human-readable parts can be recorded in the service registry, thereby allowing for the preliminary design of potential service consumer programs. Note that the Service Contract Registration process may include steps dedicated to the review of the proposed metadata.

SOA PRINCIPLES & PATTERNS

Considerations pertaining to this process are closely related to the application of the Service Discoverability (484) principle, which ensures that published service metadata is both discoverable and interpretable by a range of project team members in support of service reuse by new service consumers. Also, the pattern that represents the centralization of service metadata is Metadata Centralization [536].

Related Precepts

- Schema Design Standards
- Service Contract Design Standards
- Service-Orientation Contract Design Standards
- SLA Template

Related Roles

- Service Architect
- Technical Communications Specialist
- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Governance Specialist

NOTE

Although the Service Registry Custodian is involved in the actual implementation and governance of the service registry, this role is generally not required for governance-related tasks pertaining to the Service Contract Registration process that occurs during this stage.

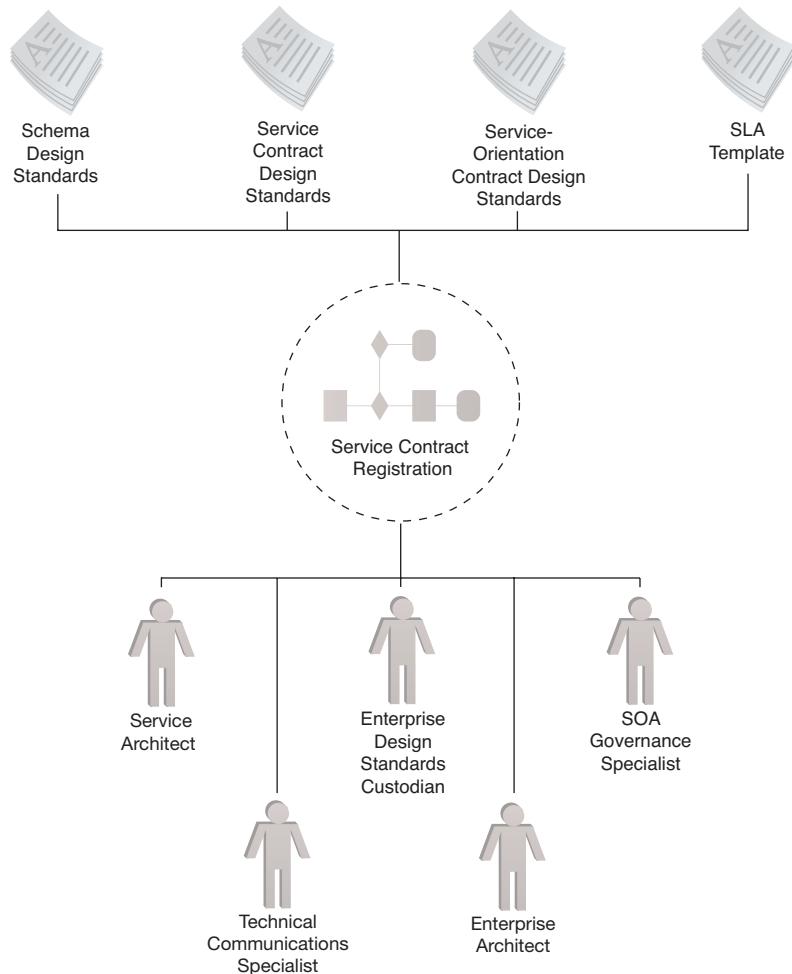


Figure 9.6

The Service Contract Registration process.

People (Roles)

Service Architect

Service-Oriented Design is of primary concern to the Service Architect, and this role's expertise supports all precepts and processes pertaining to this stage. Service Architects, together with Enterprise Design Standards Custodians, will lead the creation of design standards, as well as the subsequent compliance reviews.

Related Precepts

- Schema Design Standards
- Service Contract Design Standards
- Service-Orientation Contract Design Standards
- SLA Template

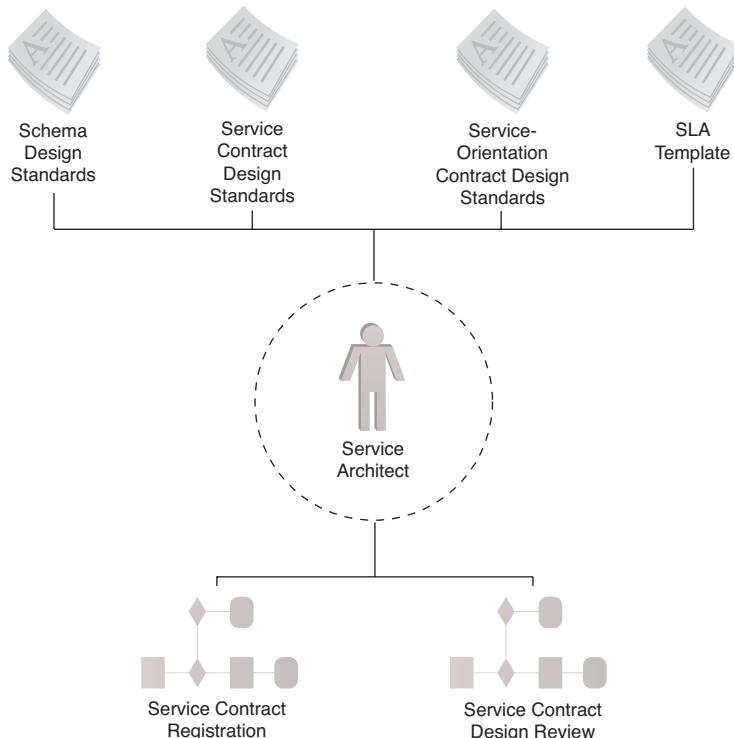


Figure 9.7

Service-Oriented Design governance precepts and processes associated with the Service Architect role.

Related Processes

- Service Contract Registration
- Service Contract Design Review

Schema Custodian

The Schema Custodian's governance responsibilities can extend to contributing to the definition of design standards pertaining to schemas and data models used by service contracts, as well as taking part in the corresponding compliance review.

SOA PRINCIPLES & PATTERNS

This role's association with the Service-Orientation Design Standards precept is specifically related to the application of the Standardized Service Contract (475) principle.

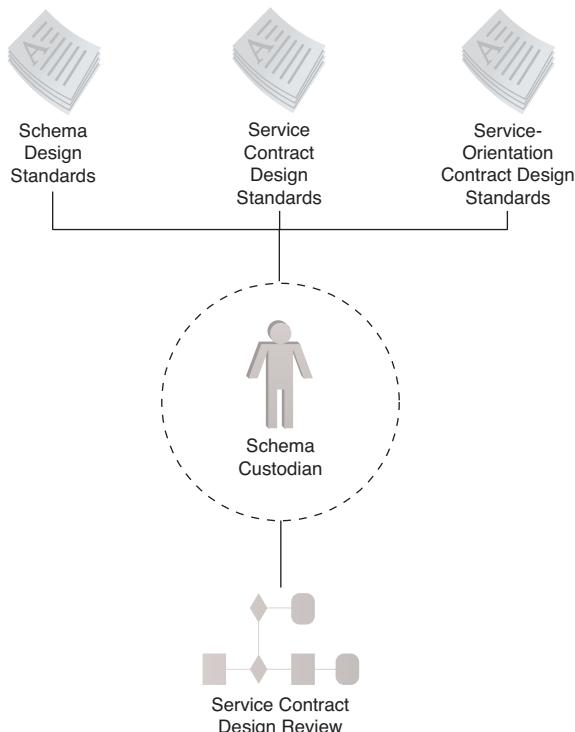


Figure 9.8

Service-Oriented Design governance precepts and processes associated with the Schema Custodian role.

When working with REST services, the Schema Custodian may further be tasked with identifying and defining the usage of service-specific and service inventory-wide media types. Specifically, the Schema Custodian may be responsible for:

- Ensuring that the most widely used and standardized media type applicable is used in the service contracts.
- Ensuring that new media types are as reusable as possible across service contracts.
- Ensuring that when new media types are defined for a given service, they are further promoted to foster discovery and reuse in support of other services within the same service inventory.

These tasks are in addition to the use of custom XML Schema definitions that may be required for data passed between REST services.

Related Precepts

- Schema Design Standards
- Service Contract Design Standards
- Service-Orientation Contract Design Standards

Related Processes

- Service Contract Design Review

Policy Custodian

As explained in Chapter 12, this role is central to precepts and processes associated with the standards and centralization of business and operational policies. Policy Custodians may further be required to share expertise for the definition of design standards for service contracts (especially with regards to the development and usage of service-specific policies).

Related Precepts

- Service Contract Design Standards

Related Processes

- Service Contract Design Review

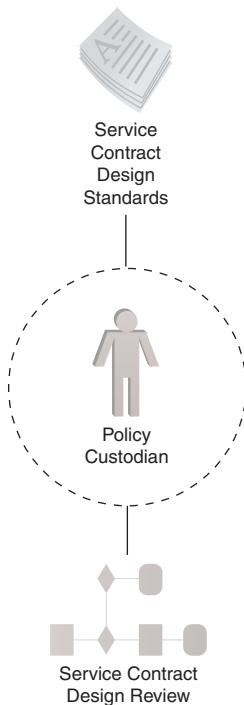


Figure 9.9
Service-Oriented Design
governance precepts and
processes associated with the
Policy Custodian role.

Technical Communications Specialist

The Technical Communications Specialist is not expected to be able to contribute governance expertise for precepts associated with the Service-Oriented Design stage. However, it may be helpful to include these specialists during review processes to ensure that service contract content (including SLA content) and associated discovery metadata being submitted is adequately clear and understandable to a range of IT professionals.

SOA PRINCIPLES & PATTERNS

During the Service Contract Design Review process, there is usually no one better qualified to check for compliance to the Service Discoverability (484) principle than the Technical Communications Specialist. Proofing the interpretability and discoverability of information published about a service also carries over into the Service Contract Registration process.

Related Precepts

- Service-Orientation Contract Design Standards

Related Processes

- Service Contract Registration
- Service Contract Design Review

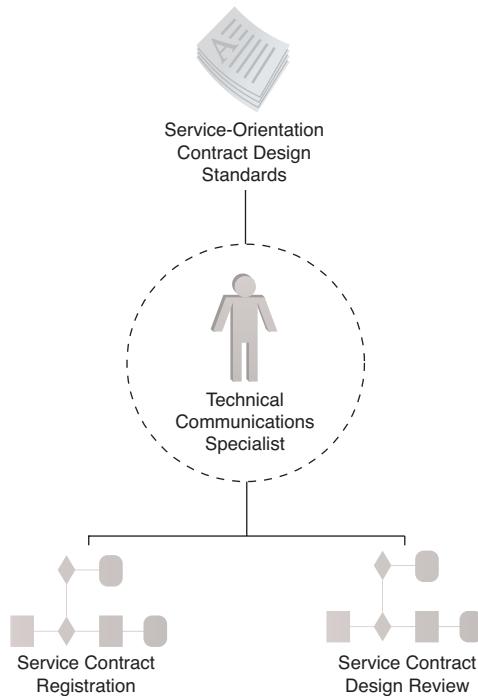


Figure 9.10

Service-Oriented Design governance precepts and processes associated with the Technical Communications Specialist role.

Enterprise Design Standards Custodian

Several of the design standards that affect and shape service contracts will be contributed by roles focused on specific subsets of the contract content or specific aspects of the contract design. For example, Schema Custodians are concerned with the schema(s) used by a service contract, whereas Policy Custodians are focused on the contract's policies. The Enterprise Design Standards Custodian collaboratively participates with these and other individuals to create the necessary design standards, but further provides a more global perspective to ensure that individual standards do not conflict with each other or with design standards that apply to other parts of the service architecture, and the service inventory architecture as a whole. This same enterprise perspective can be helpful during subsequent review processes.

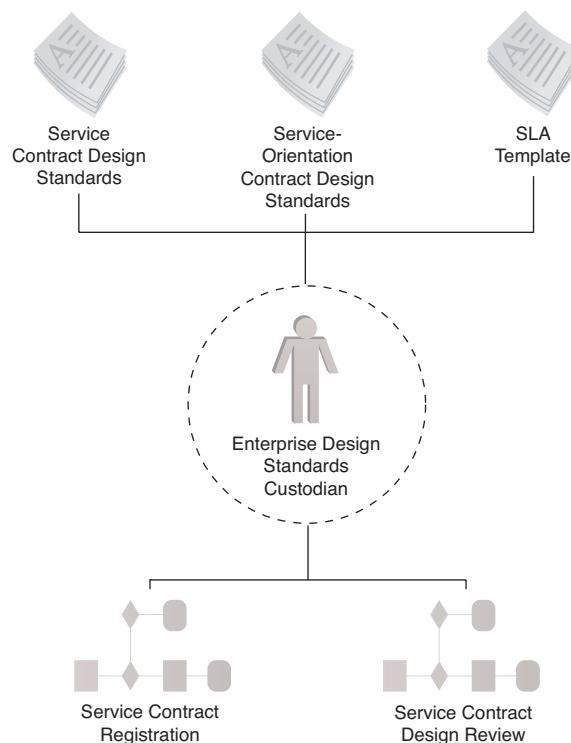


Figure 9.11

Service-Oriented Design governance precepts and processes associated with the Enterprise Design Standards Custodian role.

Additionally, the Enterprise Design Standards Custodian may be responsible for defining service contract design standards not addressed by other SOA governance-specific roles. For example, when standardizing the use of the uniform contract for REST services, the Enterprise Design Standards Custodian may need to complement media type standards provided by the Schema Custodian with further custom design standards that govern HTTP method usage and resource identifier syntax conventions.

Related Precepts

- Service Contract Design Standards
- Service-Orientation Contract Design Standards
- SLA Template

Related Processes

- Service Contract Registration
- Service Contract Design Review

Enterprise Architect

While Enterprise Architects can become involved in any area of Service-Oriented Design governance, their primary point of interest is not as much the design of the service contract, but how the service will be positioned to perform amidst other services and the enterprise platform overall. This consideration brings Enterprise Architects into the definition of the SLA Template precept, as well as the review for compliance to requirements expressed in service SLAs. This role is especially important when SLA standards need to be established for foreign or third-party cloud environments, as service quality guarantees may be tied to these cloud platforms and therefore may lie outside of the Enterprise Architects direct control.

Related Precepts

- SLA Template

Related Processes

- Service Contract Design Review
- Service Contract Registration

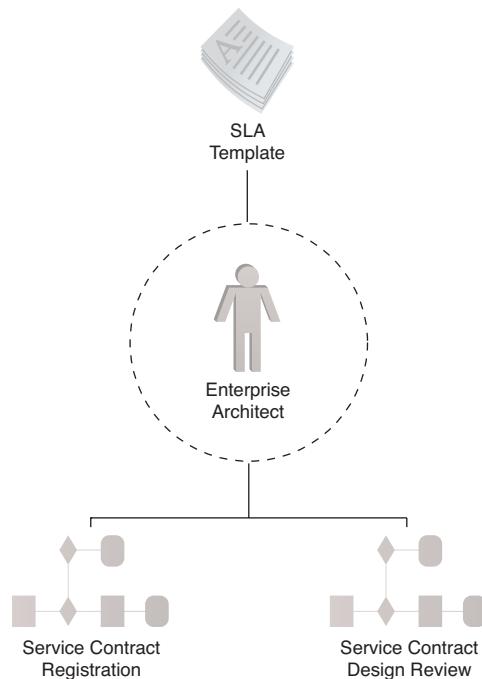


Figure 9.12

Service-Oriented Design governance precepts and processes associated with the Enterprise Architect role.

SOA Security Specialist

SOA Security Specialists can join governance groups responsible for standards definition and review when service security requirements need to be expressed within the technical interface of a service contract or as part of the service's published SLA. In this circumstance, the expertise provided by this role can assist in determining appropriate security technologies and constraints, primarily with the intention of protecting message data and transmissions.

Shared services being created with high reusability expectations will be of particular concern to SOA Security Specialists. At the service contract level there may be certain controls that need to be established to ensure that some types of potentially harmful message content cannot make it through to the service logic. Further, service contracts published for cloud-based services will likely require extra rigor in their security assessment, especially with SaaS offerings that are being made available to larger communities of service consumers.

Related Precepts

- Service Contract Design Standards
- Service-Orientation Contract Design Standards
- SLA Template

Related Processes

- Service Contract Design Review

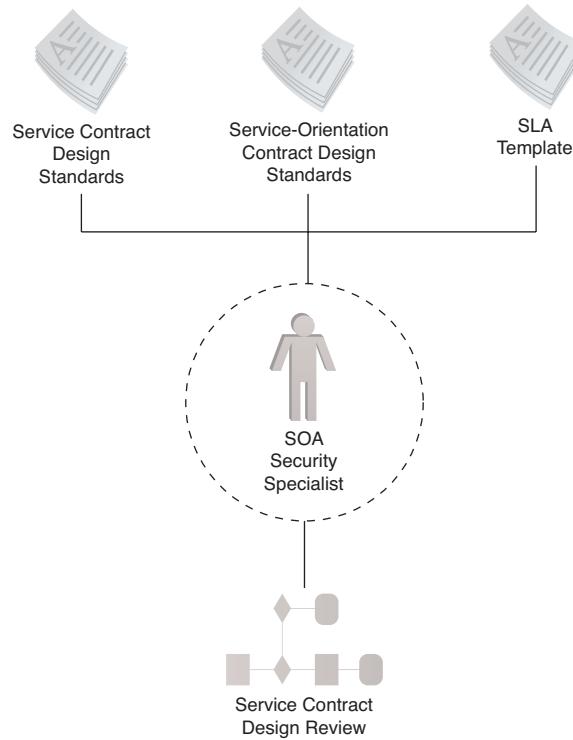
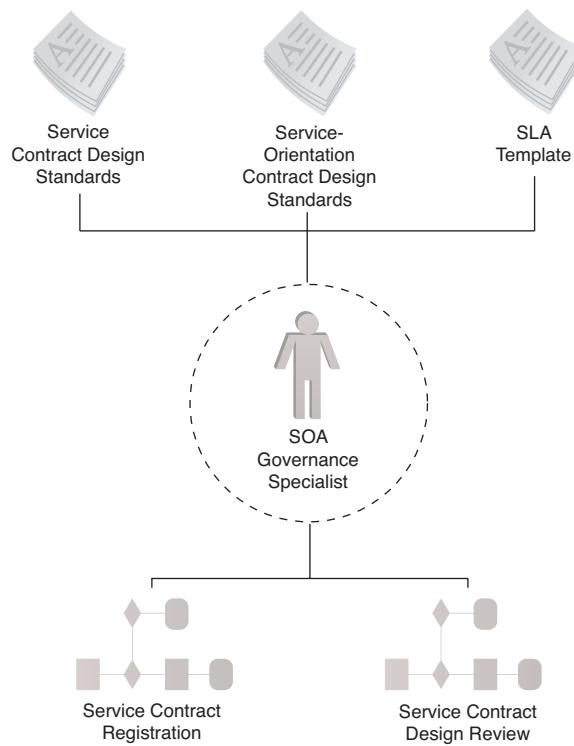


Figure 9.13

Service-Oriented Design governance precepts and processes associated with the SOA Security Specialist role.

SOA Governance Specialist

Service-Oriented Design represents the first stage within the SOA project lifecycle wherein physical service artifacts are defined and created. In addition to their regular involvement with establishing required standards, SOA Governance Specialists therefore need to assist with the coordination of other roles as well as the coordination of affected artifacts, as they transition through review processes over to the Service Logic Design stage.

**Figure 9.14**

Service-Oriented Design governance precepts and processes associated with the SOA Governance Specialist role.

Related Precepts

- Service Contract Design Standards
- Service-Orientation Contract Design Standards
- SLA Template

Related Processes

- Service Contract Registration
- Service Contract Design Review

NOTE

Service Developers can be involved in the creation and delivery of technical service contract definitions during this stage; however, they do not commonly contribute to governance tasks.

SUMMARY OF KEY POINTS

- One of the main reasons a separate project stage dedicated to the definition and creation of service contracts exists is to provide an opportunity for service contracts across a service inventory to be standardized and aligned.
- Governance precepts that relate to the Service-Oriented Design project stage help ensure endpoint federation and intrinsic interoperability among services within a service inventory.
- The Service Contract Design Review process further helps ensure standards compliance prior to proceeding to the Service Logic Design stage.

CASE STUDY EXAMPLE

One of several Service Contract Design Standard precepts within Raysmoore's SOA governance program is a custom Canonical Messaging Schema precept, which supports the Standardized Service Contract (475) principle by requiring that service contracts share the same schemas for primary business documents. The precept is explained in Table 9.1.

Canonical Messaging Schema Precept	
<p><i>Objective:</i> All services within the same service inventory must share data based on canonical data models wherever possible.</p>	
<p><i>Policy:</i> Ensure that canonical schemas are defined for shared business documents and records.</p>	<p><i>Policy:</i> Ensure that canonical schemas are sufficiently flexible to accommodate reusability by agnostic services.</p>
<p><i>Standard:</i> All canonical schemas must be defined and implemented using the XML Schema Definition Language.</p>	<p><i>Guideline:</i> Have canonical schema validation logic (in particular required fields) reviewed by Data Architects to ensure the schema validation granularity is properly balanced for reuse within the service inventory.</p>
<p><i>Standard:</i> Require that all service contracts are reviewed for compliance in accordance with the canonical schemas as part of the Service Contract Design Review process.</p>	

Table 9.1

The Canonical Messaging Schema precept.

Raysmoore's Service Architects and Data Architects collaborate to establish a canonical schema to be used by messages containing product data.

The data model is comprised of the following fields:

- location code (a unique code for each physical location within the Raysmoore corporation)
- location description
- universal SKU
- local stock number
- item short description
- QoH (Quantity on Hand)
- date (when the requested QoH is needed or is available)
- UoM (Unit of Measure)
- price (before any applicable discounts)

- discount percentage (when applied)
- inter-company billing price
- item description

All the fields in the product record are optional, and the actual amount of information depends on the details of the service being invoked.

For example:

- The Local Stock service might generate a message that includes an inventory record with only the location code and universal SKU fields completed. This service may then return one or more complete records with the inventory level of the specified item, as well as other items that are possible substitutes.
- The Report service can issue messages wherein the date field is used to determine inventory levels for a specified date range.
- The Product service uses the same basic structure but generally leaves the location code blank.
- The Stock Locations service, on the other hand, might include an inventory record containing just a universal SKU value in order to return a more complete inventory record for each warehouse location with non-zero stock of that item.

Fields containing confidential information (such as an inter-company billing price value) are normally blank, except when the service is invoked by accredited service consumers authorized to receive such information. This data structure can be extended in the future by adding new fields in compliance with the existing versioning strategy.

After a few minor corrections, the SOA Governance Program Office approves the service contract design standards and requests that the project team publish the service contracts (using this schema) to the service registry.

9.2 Governing Service Logic Design

The completion of the Service-Oriented Design stage results in the physical definition of a service's technical interface and associated definitions related to the overall service contract. Creating the corresponding design of the internal service logic requires further regulation to ensure that the underlying service architecture and associated programming and encapsulated resources can collectively fulfill the functionality promised by the service contract, while continuing to maintain required levels of cross-service consistency.

Therefore, governance efforts continue to be focused on standardization. Unlike Service-Oriented Design activities, which commonly produce actual markup code for the technical service contract, this stage is about the authoring of a design specification that will be handed over to developers for implementation. This can make the quality of the content being reviewed and assessed for compliance to governance precepts more challenging. Not only do those participating in these governance activities need to verify that a given service design specification is compliant with precepts, they must also ensure that stated compliance is realistic with regards to available resources and tools.

Precepts

Service Logic Design Standards

Service architectures and programming are subject to a range of custom design standards.

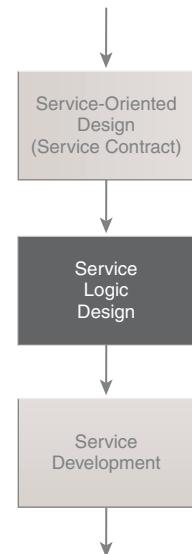
Areas that commonly require formal standardization include:

- legacy resource encapsulation
- shared or replicated resource encapsulation
- involvement of cloud computing mechanisms and other cloud-based IT resources
- composition member involvement or limitations

SOA PRINCIPLES & PATTERNS

Numerous SOA design patterns can relate to this precept. Some common examples include the Service Façade [558] and Legacy Wrapper [532] patterns.

As further explained in the upcoming *Service-Orientation Architecture Design Standards* section, several design principles also directly impact the Service Logic Design and can themselves form the basis of various design standards.



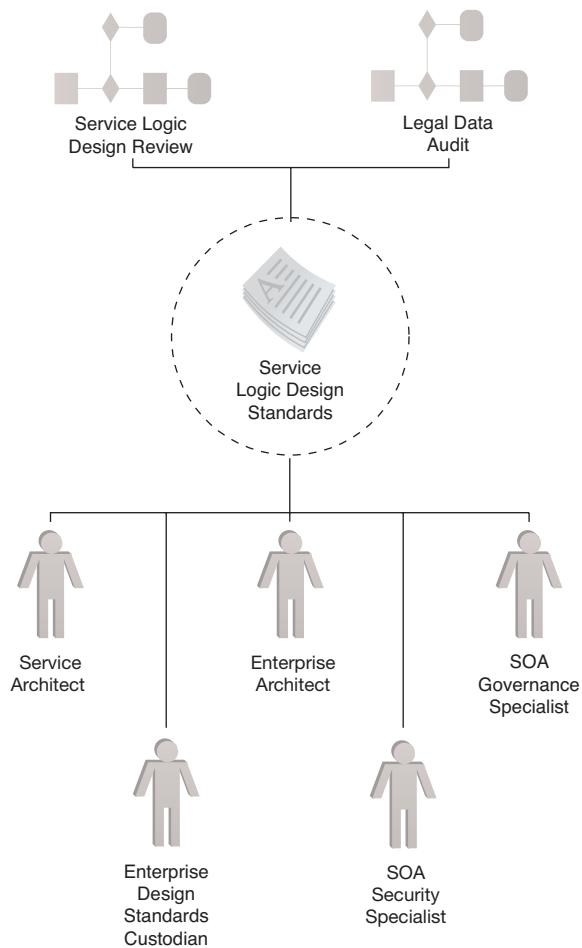


Figure 9.15

The Service Logic Design Standards precept.

- invocation of third-party or cloud-based services
- multi-tenancy requirements
- security controls and the involvement of security mechanisms
- performance and response times
- reliability, failover, and resiliency
- scalability thresholds
- data volume throughput
- REST service capabilities to defer session state back to service consumers at the end of each request
- supply of runtime metadata for REST services, such as introspection capabilities and cache directives
- the required behavior of service consumers (for example, including a virtual server to execute logic on behalf of services invoked by service consumer logic as part of the code on-demand REST constraint)

This precept is important as it demands the existence of custom design standards and avoids a common pitfall with this stage where compliance only to service-orientation design principles is considered.

Related Processes

- Service Logic Design Review
- Legal Data Audit

Related Roles

- Service Architect
- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Security Specialist
- SOA Governance Specialist

Service-Orientation Architecture Design Standards

The design of service-oriented logic requires that compliance be checked for all eight service-orientation principles. When focusing on the logic that underlies the service contract, the following principles are primarily relevant:

- Service Reusability (479)
- Service Autonomy (481)
- Service Statelessness (482)
- Service Composability (486)

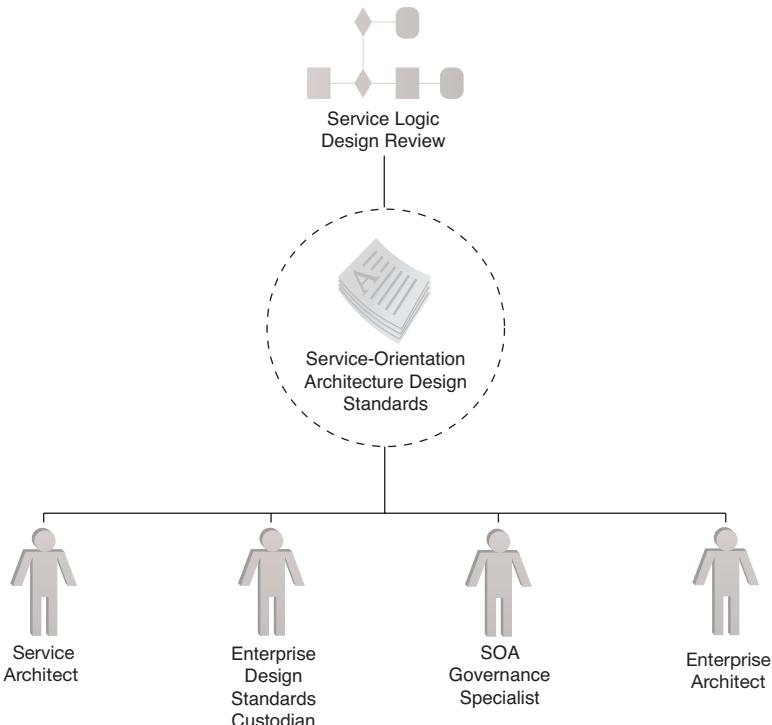


Figure 9.16

The Service-Orientation Architecture Design Standards precept.

The service contract-related principles referred to in the preceding Service-Orientation Contract Design Standards precept are revisited, especially in relation to coupling concerns. The Service Loose Coupling (477) principle in particular can play a significant role in Service Logic Design, as several negative coupling types can result from how the service logic and implementation relate to the service contract.

Related Processes

- Service Logic Design Review

Related Roles

- Service Architect
- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Governance Specialist

Processes

Service Access Control

Services are preferably delivered as “black boxes” where the only information that is published about each service is the service contract and its associated SLA, as well as service registry metadata. Details about the service architecture and its underlying implementation are hidden from project teams wanting to potentially build service consumers. This access limitation is put in place to protect those project teams from building consumer programs that inadvertently (or indirectly) form dependencies upon aspects of the underlying service implementation that may be subject to change.

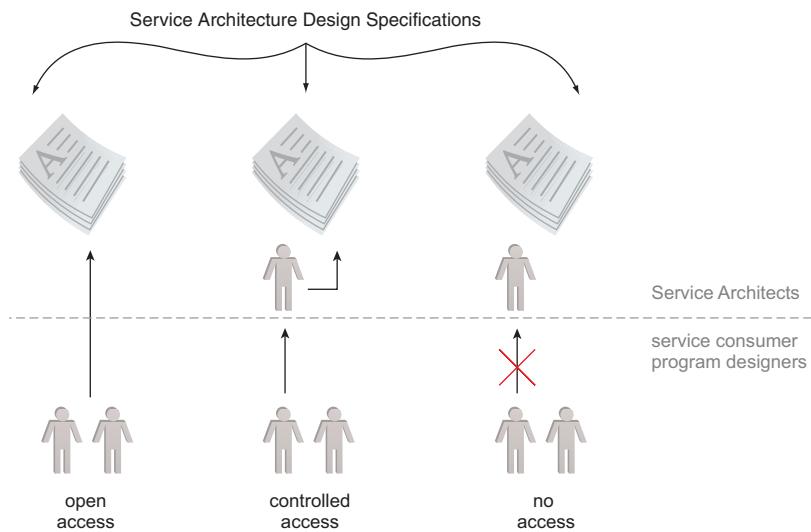
As a result, a process needs to be established to control access to private service design specifications. Figure 9.17 depicts different levels of access service consumer designers can be granted. Although shown, the open access option is not usually a viable option as it does not introduce any level of access control.

Related Precepts

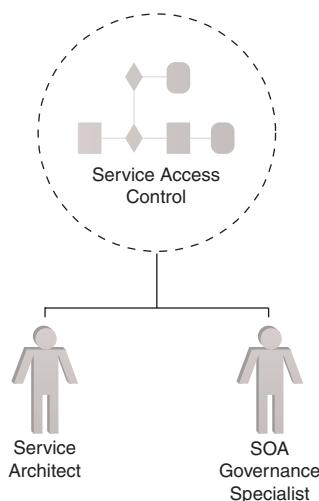
N/A

Related Roles

- Service Architect
- SOA Governance Specialist

**Figure 9.17**

Higher levels of access control allow for abstraction levels to be consistently preserved during the lifetime of a service. (This figure is borrowed from Chapter 8 of the *SOA Principles of Service Design* book.)

**Figure 9.18**

The Service Access Control process.

Service Logic Design Review

This process establishes a series of steps during which qualified team members assess the Service Logic Design specification to validate compliance with both the custom design standards and service-orientation design principles. Additionally, this process revisits the service contract information in relation to the Service Logic Design to ensure alignment and to confirm that all documented parts of the design completely and accurately provision the functionality expressed in the service contract.

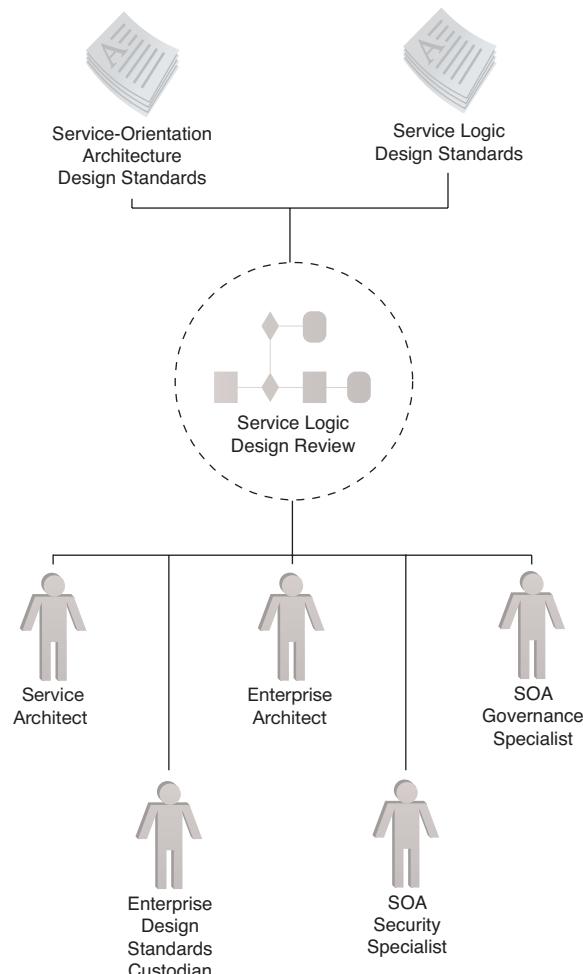


Figure 9.19

The Service Logic Design Review process.

Because a large part of governing compliance to Service Logic Design Standards involves verification that preceding analysis and design steps have been performed successfully, use of checklists is recommended.

For example, checklists can be created to verify that a service:

- conforms to the interface and SLA terms specified in the published service contract
- supports known functional requirements
- supports non-functional requirements and security requirements
- has been subject to the application of all relevant and required design patterns
- is addressing required levels of runtime quality and behavioral consistency
- can or must be deployed in a cloud environment (or, alternatively, cannot be cloud-based)
- is using optimal and authorized technologies
- is correctly implementing identified business rules

The main value of a checklist is that it requires a physical or electronic signature of an approver or set of approvers, creating an audit trail of responsibility in the event of later inquiries.

NOTE

Although related to the Service Testing stage, a common deliverable required as part of the Service Logic Design stage exit criteria is a test plan containing functional and non-functional areas for which the service should be tested, as derived from the Service Logic Design specification and the review results.

For any services where the design is taking an excessive amount of effort, or where one or more checklists cannot be completed satisfactorily, those involved with the Service Logic Design Review can take extra steps to identify the root cause of the problem to provide guidance in support of any necessary remedial action.

Related Precepts

- Service Logic Design Standards
- Service-Orientation Architecture Design Standards

Related Roles

- Service Architect
- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Security Specialist
- SOA Governance Specialist

Legal Data Audit

There may be occasions where certain types of data processed by the service logic have legal requirements.

Examples include:

- The data has privacy requirements that must be fulfilled by appropriate security controls and logic.
- The data has geographical requirements that may limit options for the service and/or the data repository from being deployed in certain types of cloud environments and remote data centers.
- The data may be subject to legal policies prone to change, which may limit or altogether eliminate the possibility of deploying the service and/or the data in a cloud environment that can impose mobility restrictions.

The Legal Data Audit precept establishes a process whereby data accessed, stored, or processed in relation to the service logic is checked against any known or identified legal requirements. This can be a potentially involved process when cloud services (in particular cloud storage services) are being reviewed—or—when cloud services or cloud storage repositories are being invoked or accessed by the service being reviewed. In the latter case, a previously existing service may never have had a legal compliance issue until made part of a new service composition.

Related Precepts

- Service Logic Design Standards

Related Roles

- Service Architect
- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Security Specialist
- SOA Governance Specialist
- Other: Data Architect

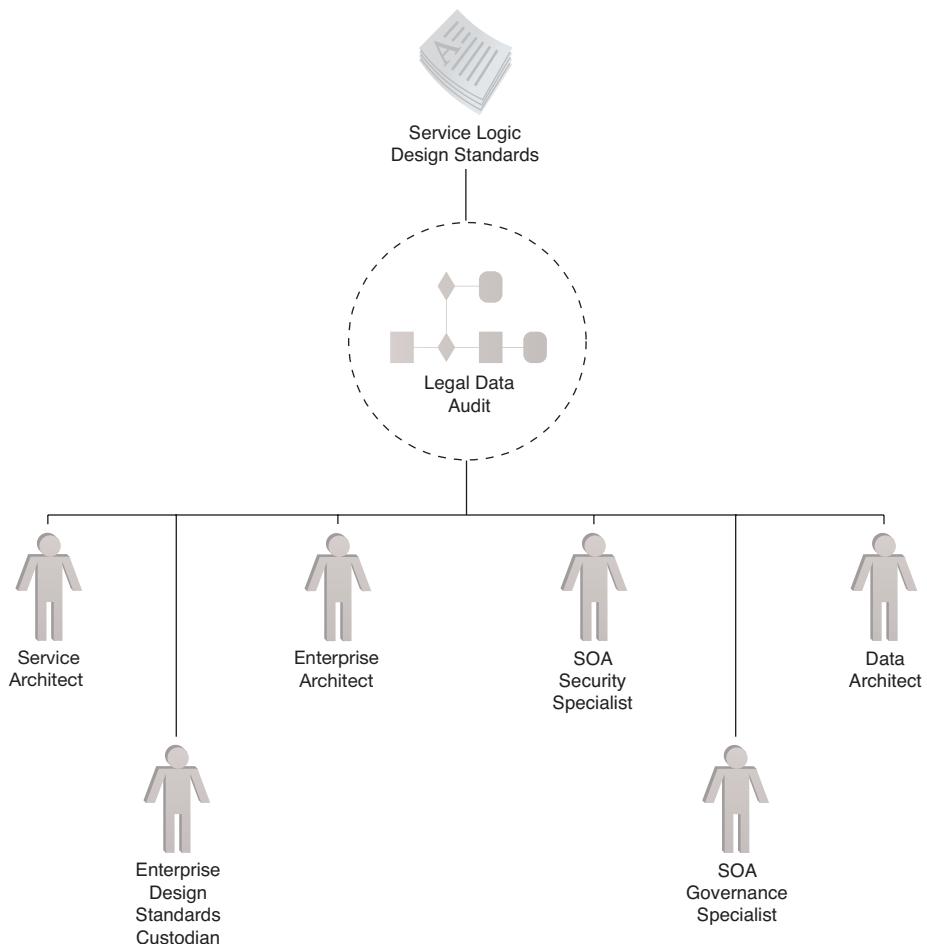


Figure 9.20

The Legal Data Audit process.

People (Roles)

Service Architect

Service Logic Design is all about defining the service architecture, from the service contract all the way down to the processing layers that interact with platform resources, such as databases, legacy applications, and other services. This makes governance tasks pertaining to this stage a focal point for Service Architects. This role tends to participate in all covered areas of Service Logic Design governance.

Related Precepts

- Service Logic Design Standards
- Service-Orientation Architecture Design Standards

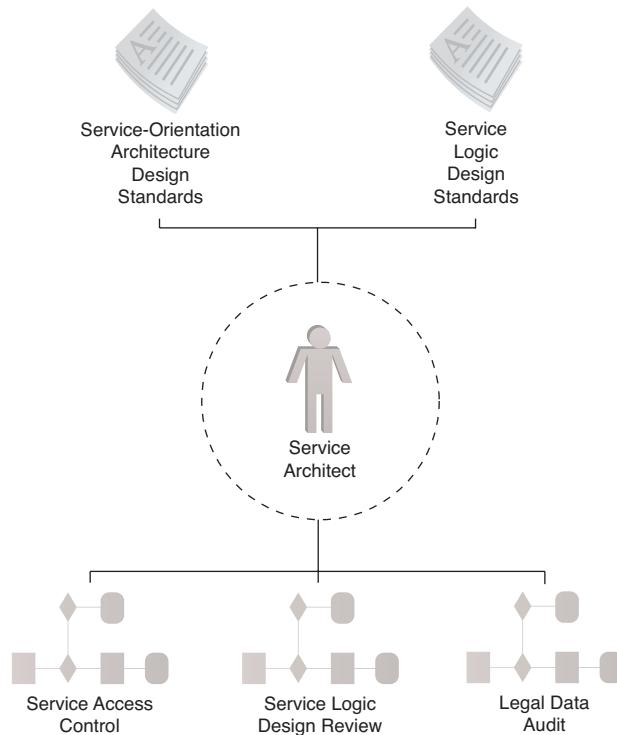


Figure 9.21

Service Logic Design governance precepts and processes associated with the Service Architect role.

Related Processes

- Service Access Control
- Service Logic Design Review
- Legal Data Audit

Enterprise Design Standards Custodian

Service Architects will typically team up with the Enterprise Design Standards Custodian to finalize proposed service logic and architecture design standards, some of which can be influenced by legal requirements. This role will further participate in the subsequent review.

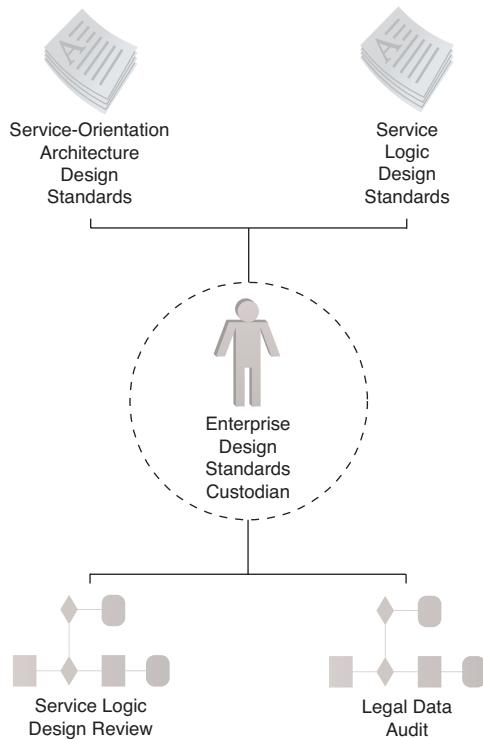


Figure 9.22

Service Logic Design governance precepts and processes associated with the Enterprise Design Standards Custodian role.

Related Precepts

- Service Logic Design Standards
- Service-Orientation Architecture Design Standards

Related Processes

- Service Logic Design Review
- Legal Data Audit

Enterprise Architect

How the Enterprise Architect takes part in Service Logic Design governance activities can vary greatly, mostly dependent on the nature of a given service architecture. For example, autonomous on-premise service architectures may require less governance attention from enterprise architectures than those that are cloud-based or require access to a great deal of shared enterprise resources, such as central databases. Either way, it

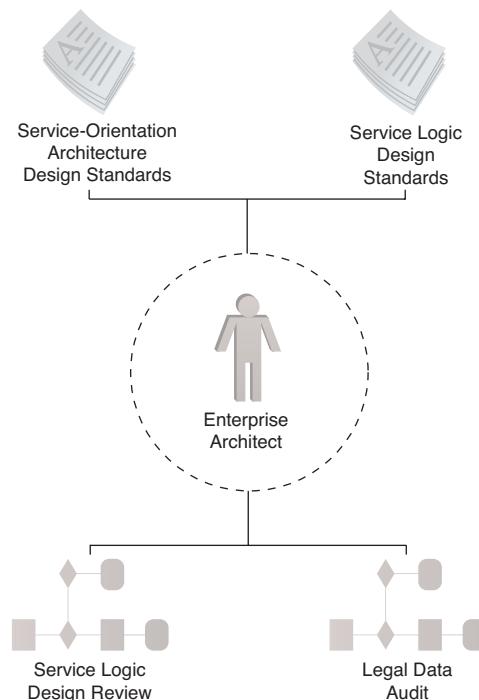


Figure 9.23

Service Logic Design governance precepts and processes associated with the Enterprise Architect role.

is common for the Enterprise Architect to be part of the actual Service Logic Design Review process and for this person's sign-off to be required as part of the exit criteria.

Related Precepts

- Service Logic Design Standards
- Service-Orientation Architecture Design Standards

Related Processes

- Service Logic Design Review
- Legal Data Audit

SOA Security Specialist

A given service architecture specification can include preventative and reactive security processing logic and the use of external security mechanisms. The definition of security-related design standards and their approval will generally require the involvement of an SOA Security Specialist.

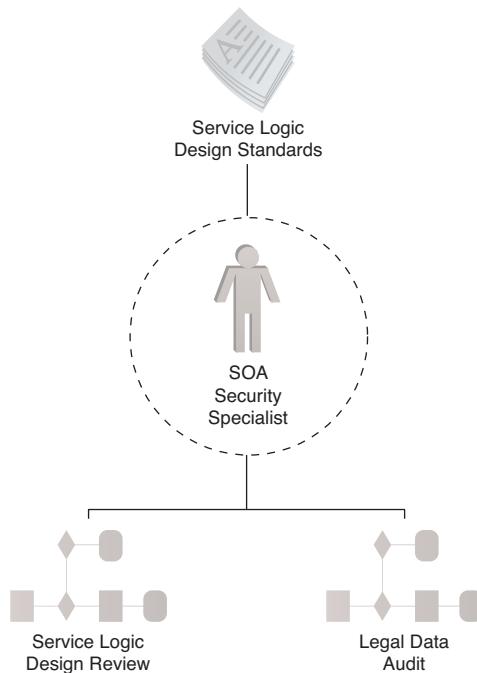


Figure 9.24

Service Logic Design governance precepts and processes associated with the SOA Security Specialist role.

As explained in Chapter 7, depending on the extent of security requirements and the level of importance assigned to fulfilling these requirements, a separate Security Audit process may be needed (especially when there are security requirements with legal implications). Such a process would be led by the individual or group taking on this role.

Related Precepts

- Service Logic Design Standards

Related Processes

- Service Logic Design Review
- Legal Data Audit

SOA Governance Specialist

The exit criteria for the Service Logic Design is especially important to the SOA Governance Specialist, because it represents the final step before the service moves on to Service Development, where it will be actually created. Further, the SOA Governance Specialist will be responsible for overseeing (and possibly carrying forward) Service Logic Design precepts in relation to precepts from previous stages, in particular the Service-Oriented Design stage.

For example, issues may arise during the Service Logic Design Review that end up impacting the design of the service contract. This may require that previously applied precepts be revisited or that a waiver may be issued or that the Service Contract Design Review process be carried out again. The SOA Governance Specialist can provide guidance to help ensure that these and other necessary governance responsibilities are carried out in preparation for Service Development.

Finally, the SOA Governance Specialist can assist the Service Architect with establishing the Service Access Control process, as this type of regulatory procedure may have various organizational governance implications.

Related Precepts

- Service Logic Design Standards
- Service-Orientation Architecture Design Standards

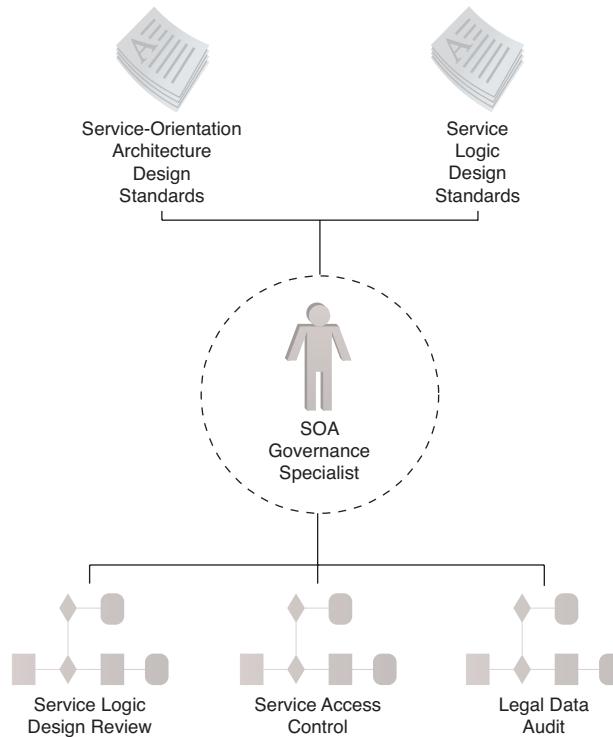


Figure 9.25

Service Logic Design governance precepts and processes associated with the SOA Governance Specialist role.

Related Processes

- Service Access Control
- Service Logic Design Review
- Legal Data Audit

NOTE

Policy Custodians are not typically involved in governance tasks during this stage, unless specific policy processing requirements (internal to the service architecture) exist that warrant governance attention.

SUMMARY OF KEY POINTS

- The Service Logic Design Review is the fundamental means by which the Service Logic Design stage is governed.
 - In addition to adhering to custom design standards, service designs need to be compliant to service-orientation design principles. This ensures foundational, baseline consistency across all services in support of the strategic goals of service-oriented computing.
-

CASE STUDY EXAMPLE

The SOA Governance Program Office creates the following set of checklists in support of the Service Logic Design Standards and Compliance with Service-Orientation Architecture Design Principles precepts:

- A *service functionality checklist* to validate that each service design fully meets its business requirements.
- A *service non-functional requirements checklist* to validate that each service design fully supports its performance and behavioral requirements.
- A *service security checklist* to validate that the service design fully meets the security requirements.
- A *service architecture checklist* to validate that the service design meets all architectural standards in compliance with the overall service inventory architecture.

Using their service registry, the SOA Governance Program Office further automates the distribution of these checklists to authorized reviewers and approvers in order to implement a policy that prevents the Service Development stage from beginning until all checklists have been formally approved and signed off.

When the Service Custodian responsible for the Product service submits this service to the Service Logic Design Review, she raises a concern regarding one of the non-functional requirements on the SOA Governance Program Office checklists. The requirement corresponds to a design standard produced by the Enterprise Design Standards Custodian that demands that all service capabilities respond within a maximum period of 2 seconds. Depending on the criteria submitted, the GetRange capability of the Product service may need to query multiple product inventory

locations, some of which are geographically distributed. Although these types of queries are expected to be rare, the fact that they are possible means that the upcoming Service Testing stage will likely reveal this as an area of non-compliance.

The SOA Governance Program Office joins the Service Custodian and a group of Service Architects to discuss the issue. During the meeting they identify four possible options:

1. Use a database replication approach to maintain a centralized “super table” that contains a copy of all product inventory data across Raysmoore and its subsidiaries. The Product service would then only need to query this one table, resulting in improved performance.
2. Create additional logic that detects when geographically separated product inventory repositories need to be queried, and then respond with a message to the service consumer warning that the query will take extra time (and asking for further confirmation).
3. Do nothing until performance test results can be obtained to determine actual service response times.
4. Issue a waiver for the Product service’s GetRange capability so that the service can proceed with an approved extent of non-compliance.

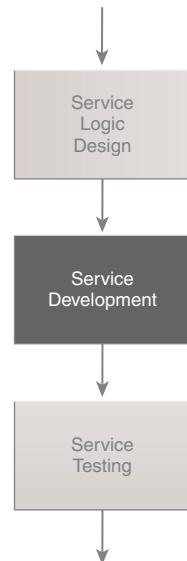
It is decided that the cost of Option #1 will be too high considering the infrequency with which this performance lag will occur. Option #2 is also not chosen, as it would have too much of an impact on service consumer requirements. Although some voted for Option #3, it was Option #4 that was finally selected with one condition: the service logic will need to be further enhanced so that a query will timeout after 5 seconds, in which case the service capability will return an error back to the service consumer.

The design of the service is updated to reflect this change, and all other checklist items are signed off, allowing the Product service to be released to development.

9.3 Governing Service Development

The development of service logic can be challenging to formally govern, especially when IT enterprises are accustomed to outsourcing development projects or to giving internal development teams a great deal of autonomy. Even within standardized SOA delivery projects, there is sometimes a tendency to grant developers creative freedom as to how a given service should be programmed, as long as the logic created conforms to approved design specifications.

Governance precepts for this stage are not intended to constrict or negatively inhibit this creativity; their purpose is to regulate development activities and approaches in support of consistency, standardization, and quality maintenance.



Precepts

Service Logic Programming Standards

Various programming language conventions can be established to ensure consistency across services.

Some common considerations include:

- naming standards
- consistent documentation and comments
- in-line maintenance logs identifying the developers that performed work on a given part of the service logic code
- reusable routines, variable names, use of global and shared variables, etc.

These types of conventions are well-established as programming best practices. However, their importance within SOA projects is amplified due to the potential need for shared services to be reused, recomposed, and perhaps maintained and expanded by different project teams at different points in time.

Furthermore, the positioning and use of industry standards will be dictated in order to ensure that proprietary programming languages are not employed when industry standardization is required. In this regard, there will tend to be overlap with the aforementioned service logic and service contract design standards. Regardless of how or where the programming standards are documented, it is important that they be made available specifically to Service Developers.

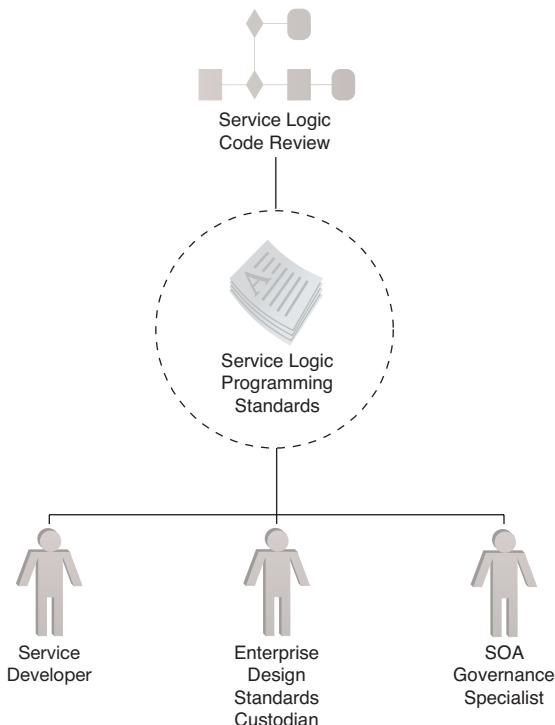


Figure 9.26

The Service Logic Programming Standards precept.

Related Processes

- Service Logic Code Review

Related Roles

- Service Developer
- Enterprise Design Standards Custodian
- SOA Governance Specialist

Custom Development Technology Standards

Existing enterprise and design standards will commonly dictate which programming languages are allowed. This will often be based on the current development platform (Java, .NET, etc.) already established within the IT enterprise, but can also be influenced by new and unique business requirements.

Also, the use of industry standards (such as XML-based markup languages) will be identified in the design specifications and further regulated (and elaborated) by custom programming standards. For example, whereas a design specification may indicate the WS-Security standard is to be used, the Custom Development Technology Standards document will identify exactly which features and elements of the standard are actually allowed and under what circumstances they may not be allowed.

Tool standards are also common to ensure that the building and subsequent maintenance processes are not bound to disparate tools and skill-sets. Having a common set of tools used throughout development-related project phases makes it easier for different developers to work with each other's code. The choice of tools will depend on the development platform and industry standards dictated by the Service Logic Programming Standards.

NOTE

One of the strategic goals of service-oriented computing explained in Chapter 3 was Increased Vendor Diversity Options. It may seem contrary to this goal to support the standardization of development tools and technologies. This precept can be augmented if the need to diversify emerges and is sufficiently justified. This is why the goal emphasizes having the “option” of diversifying (when it makes sense to do so), but it does not advocate unwarranted diversification.

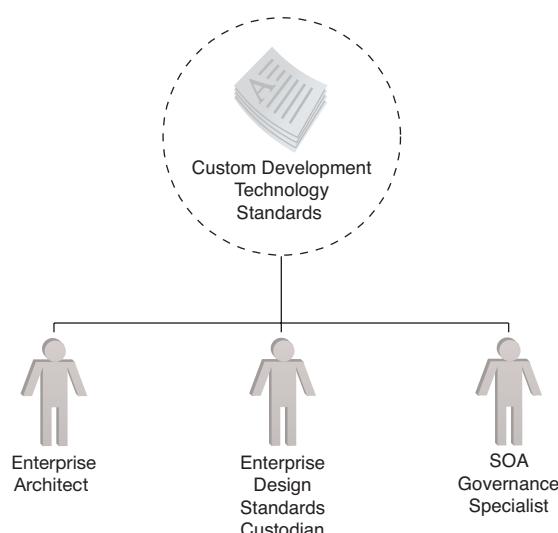


Figure 9.27

The Custom Development Technology Standards precept.

Related Processes

N/A

Related Roles

- Enterprise Design Standards Custodian
- Enterprise Architect
- SOA Governance Specialist

Processes

Service Logic Code Review

The primary purpose of the Service Logic Code Review is to verify that the programming code created and used by the service is in compliance with the Service Logic Programming Standards. The extent to which this review needs to delve into the specifics of the code is directly related to the level of detail of the programming standards. For example, some standards may dictate how certain types of routines need to be programmed, whereas others may require that existing generic routines be reused for specific types of programming requirements.

When carrying out this process, it is very common to incorporate additional types of reviews that go beyond the Service Logic Programming Standards. Examples of typical areas to further proof include:

- the service is programmed according to its design specification
- the quality of the programming code
- the level of documentation is adequate to enable future maintenance by different Service Developers

A Service Logic Code Review typically consists of structured code walkthroughs, where a group of peer developers inspects and constructively criticizes the code in sessions moderated by an SOA Governance Specialist. These walkthroughs can also help mentor junior programmers new to the use of the custom standards.

Possible outcomes of the review include:

- The code is approved as it stands.
- Potential improvement areas are identified and the Service Developer is tasked to incorporate them.

- The code is so non-compliant to the Service Logic Programming Standards that it is rejected and assigned to new Service Developers.

When the final approval is received, the service state in the registry should be updated to indicate its promotion to the next stage. Changes to the service metadata made during coding can also be reflected in the repository.

Related Precepts

- Service Logic Programming Standards

Related Roles

- Service Developer
- Enterprise Design Standards Custodian
- SOA Governance Specialist

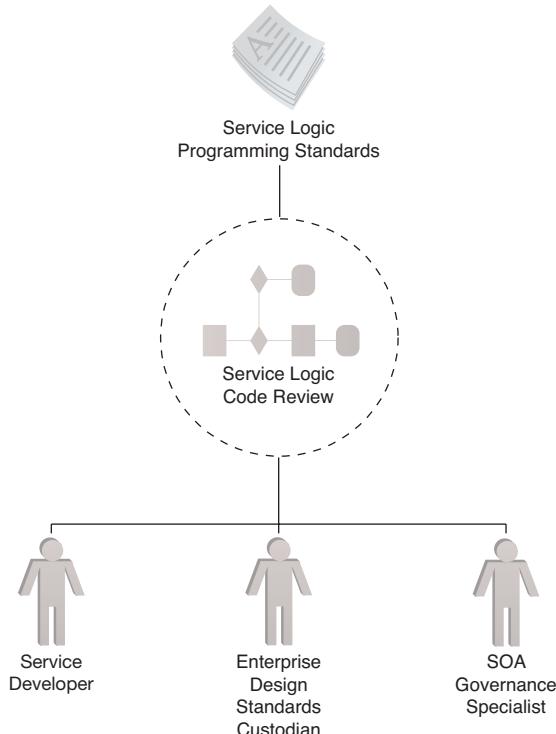


Figure 9.28

The Service Logic Code Review process.

People (Roles)

Service Developer

Although Service Developers will be the ones carrying out the programming activities during the Service Development stage, their expertise is useful and often necessary to help establish suitable Service Logic Programming Standards, especially in relation to the distinct requirements introduced by the application of service-orientation design principles (as will have likely been documented in the design specifications from the Service Logic Design stage). Service Logic Code Reviews will also require participation by peer Service Developers to verify compliance to the corresponding Service Logic Programming Standards.

Related Precepts

- Service Logic Programming Standards

Related Processes

- Service Logic Code Review

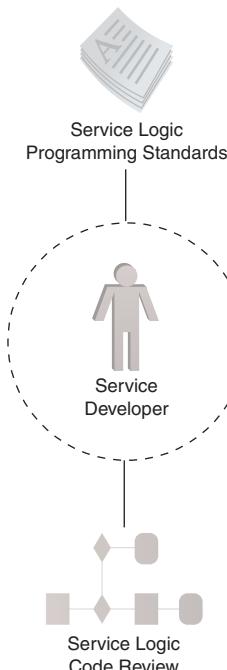


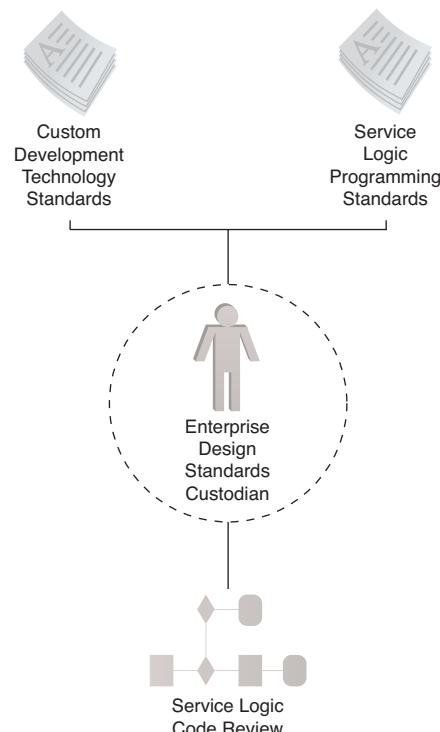
Figure 9.29

Service Development governance precepts and processes associated with the Service Developer role.

Enterprise Design Standards Custodian

This role will generally act in a peripheral capacity with regards to the definition of Service Logic Programming Standards. Similarly, involvement in the Service Logic Code Review may also be secondary to the Service Developers actually performing the detailed code audits.

The Enterprise Design Standards Custodian may communicate, in advance, specific regulations or requirements of importance to the Service Developers authoring the standards and performing the reviews. The Enterprise Design Standards Custodian may then need to rely on confirmation from the Service Developers that those requirements have been met. Upon receiving confirmation (perhaps in the form of a signature), the Enterprise Design Standards Custodian can sign off on the standards or review.

**Figure 9.30**

Service Development governance precepts and processes associated with the Enterprise Design Standards Custodian role.

This role is further involved with the definition of Custom Development Technology Standards, for which collaboration with the Enterprise Architect and other technology experts may be required.

Related Precepts

- Service Logic Programming Standards
- Custom Development Technology Standards

Related Processes

- Service Logic Code Review

Enterprise Architect

Enterprise Architects will usually have a regulatory interest in the technologies and tools being used to build software programs for deployment within the enterprise. They will therefore commonly lead the creation of Custom Development Technology Standards in cooperation with the Enterprise Design Standards Custodian and an SOA Governance Specialist.

Related Precepts

- Custom Development Technology Standards

Related Processes

N/A

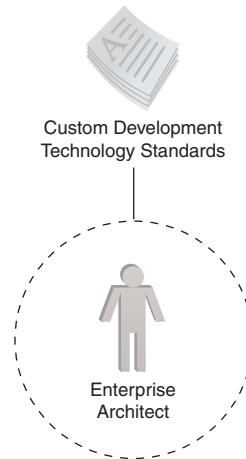


Figure 9.31

Service Development governance precepts and processes associated with the Enterprise Architect role.

SOA Governance Specialist

During the Service Development stage, SOA Governance Specialists provide any necessary guidance for the definition and application of precepts and processes on behalf of the SOA Governance Program Office.

Together with the Enterprise Design Standards Custodian, they may further assist with the mapping of development standards to design standards to ensure consistency.

Related Precepts

- Service Logic Programming Standards
- Custom Development Technology Standards

Related Processes

- Service Logic Code Review

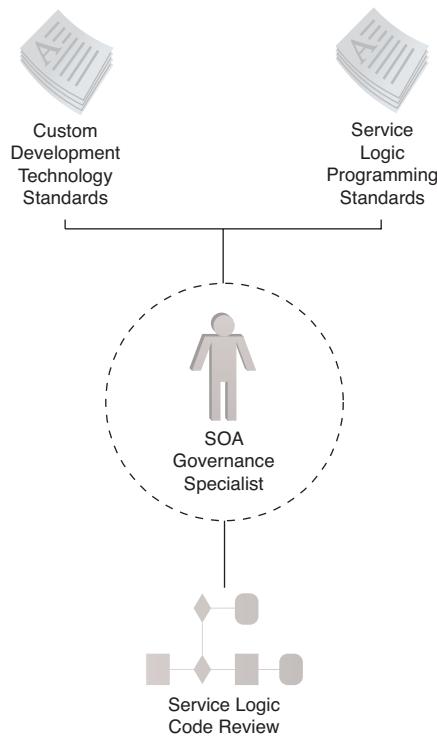


Figure 9.32

Service Development governance precepts and processes associated with the SOA Governance Specialist role.

SUMMARY OF KEY POINTS

- The focal point of the Service Development stage, from a governance perspective, is the use of development standards and the involvement of a review for checking on the compliance of those standards.
- Service Development can be a challenging stage to effectively govern because developers may not be accustomed to the form of regulation required by SOA governance precepts.

CASE STUDY EXAMPLE

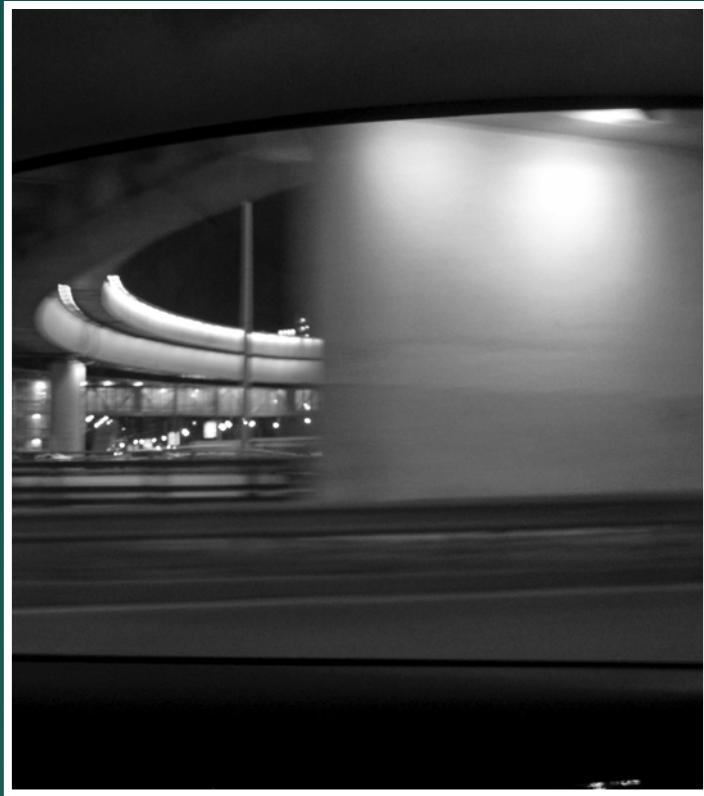
The SOA Governance Program Office mandates code walkthroughs for every newly developed or updated service. During these sessions Service Developers join Service Architects to compare the programming logic with the functionality defined in the corresponding design specifications.

During the code walkthrough for the Product service a Service Architect notices that it contains several complex routines that seem to include superfluous code not relevant to the nature of the service logic. The Service Architect asks the newly hired Service Developer responsible for these routines about their origin and, when pressed on that point, the developer admits that some code routines from a project he worked on for a previous employer had been reused.

The SOA Governance Program Office representative joins the Service Architect and they have a counseling session with the Service Developer to explain that not only have these actions caused the Product service to fail its code review, but that the inclusion of code routines from another company may have violated that company's copyright and perhaps even introduced a security breach.

The Product service logic is handed back to the development team where it will undergo rework to replace the foreign routines.

Chapter 10



Governing Service Testing and Deployment Stages

10.1 Governing Service Testing

10.2 Governing Service Deployment and Maintenance

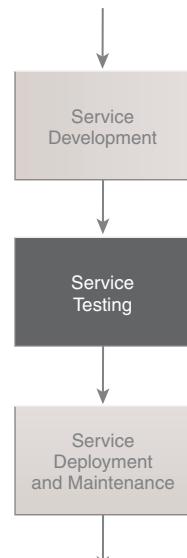
The methodology and management system used by an IT enterprise for the analysis, design, and development of services can vary based on strategic and tactical priorities and business goals. However, approaches and procedures that oversee the testing and deployment of software programs are often well established, regardless of how the programs were created. Although service delivery does introduce unique testing and deployment requirements, the same tendency to follow consistent procedures holds true. From a governance perspective, we are concerned with maintaining this consistency, especially across project teams that may be delivering different services at different times into the same service inventory.

10.1 Governing Service Testing

Service testing efforts aim to ensure compliance with functional and non-functional requirements, as well as related policies, SLAs, and other pre-defined and measurable features and behaviors.

Common types of tests and test cycles include:

- *Unit Tests* (service is tested to ensure that it performs as expected)
- *Standards Compliance Tests* (service is tested to ensure it complies with required standards)
- *Functional Tests* (service is tested to ensure it adheres to functional requirements)
- *Security Tests* (service is tested to ensure it adheres to security requirements)
- *Policy Tests* (service is tested for compliance to service-specific and cross-service policies)
- *Regression Tests* (service is tested to ensure that changes do not negatively impact existing service consumers)



- *Performance Tests* (service is tested to ensure that it adheres to non-functional requirements, including performance, SLAs, and scalability)
- *Integration Tests* (service is tested within a new or foreign environment to identify any compatibility issues and to ensure it behaves and performs as expected)

The last item on this list is discussed specifically in relation to cloud environments in this chapter.

Precepts

Testing Tool Standards

Several of the previously listed tests can be automated and executed continuously throughout the service development lifecycle. In order to maintain consistency across services and service and solution delivery projects, a common set of testing tools should be used.

Establishing Testing Tool Standards helps avoid different services within the same service inventory from being subjected to the same tests with variance in test method, test quality, and the metrics used to measure test results. The last consideration is especially important in support of the Test Parameters Standards precept (explained next).

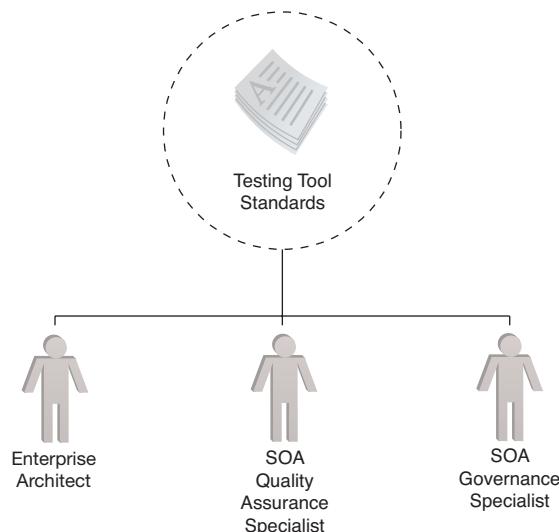


Figure 10.1

The Testing Tools Standards precept.

Related Processes

N/A

Related Roles

- Enterprise Architect
- SOA Quality Assurance Specialist
- SOA Governance Specialist

Testing Parameter Standards

Many IT departments define standards that specify which tests need to be performed along with common sets of testing parameters that services (especially shared services) must be required to meet in order to be approved for deployment. This type of service inventory-wide standardization is important to ensure behavioral predictability among services.

When certain parameters cannot be met (perhaps due to limitations imposed by legacy resources encapsulated by the service), exceptions may be allowed. In this case, there is typically a requirement to document these limitations in the service's SLA.

Related Processes

- Service Test Results Review

Related Roles

- Service Administrator
- Cloud Resource Administrator
- SOA Security Specialist
- SOA Quality Assurance Specialist
- SOA Governance Specialist

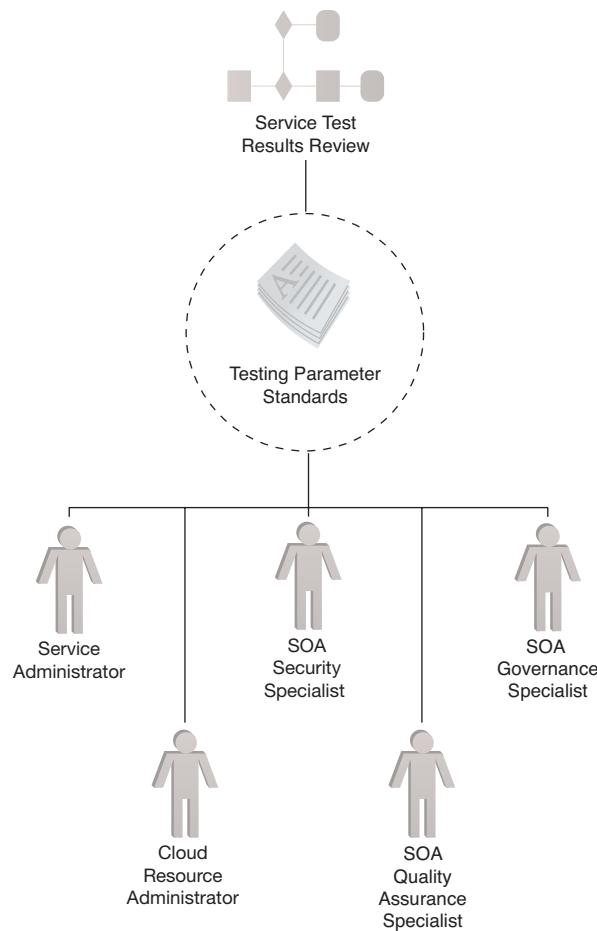


Figure 10.2
The Testing Parameter Standards precept.

Service Testing Standards

Each service architecture is unique and testing processes are commonly based on the specific set of functions and processing requirements of a given service. However, there are baseline standards that can be established to ensure a minimum guarantee of runtime performance and behavior. These base Service Testing Standards often correlate to the Runtime Service Usage Thresholds precept (and associated metrics) documented in the *Governing Service Usage and Monitoring* section in Chapter 11.

Further, this precept can encompass additional testing standards specific to different service models (task, entity, utility, etc.), as each will tend to have specific types of processing responsibilities and performance expectations.

NOTE

Due to the variance in functionality across different services and service models, some parts of this precept may be better suited as guidelines. It is therefore useful to make a clear distinction between voluntary testing guidelines and mandatory, baseline testing standards.

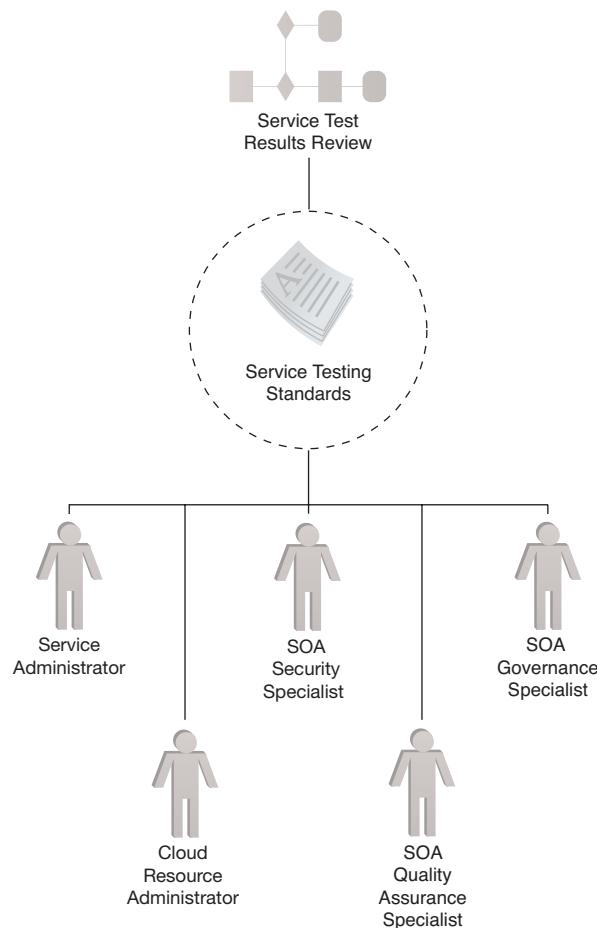


Figure 10.3

The Service Testing Standards precept.

Related Processes

- Service Test Results Review

Related Roles

- Service Administrator
- Cloud Resource Administrator
- SOA Security Specialist
- SOA Quality Assurance Specialist
- SOA Governance Specialist

Cloud Integration Testing Standards

For services identified for deployment within third-party cloud platforms, further base-line testing standards can be defined to guarantee that services behave and perform to a level of acceptable consistency with on-premise services. This is especially important when service compositions can span cloud and on-premise deployed services.

Cloud Integration Testing Standards include criteria and checklists that confirm compatibility with specific implementation characteristics required to maintain cross-service standardization. The identification of non-compliant (and non-flexible) platform requirements can lead to the rejection of the cloud as a deployment option during the subsequent Service Test Results Review.

Conversely, a primary motivation for targeting the deployment of a service within a cloud platform may be the availability of IT resources (especially in relation to dynamic scaling and pay-per-use billing) that do not exist within the IT enterprise's on-premise environment. In this case, cloud integration tests may be further extended to validate that the cloud-specific features do, in fact, address the service requirements and expectations.

Related Processes

- Service Test Results Review

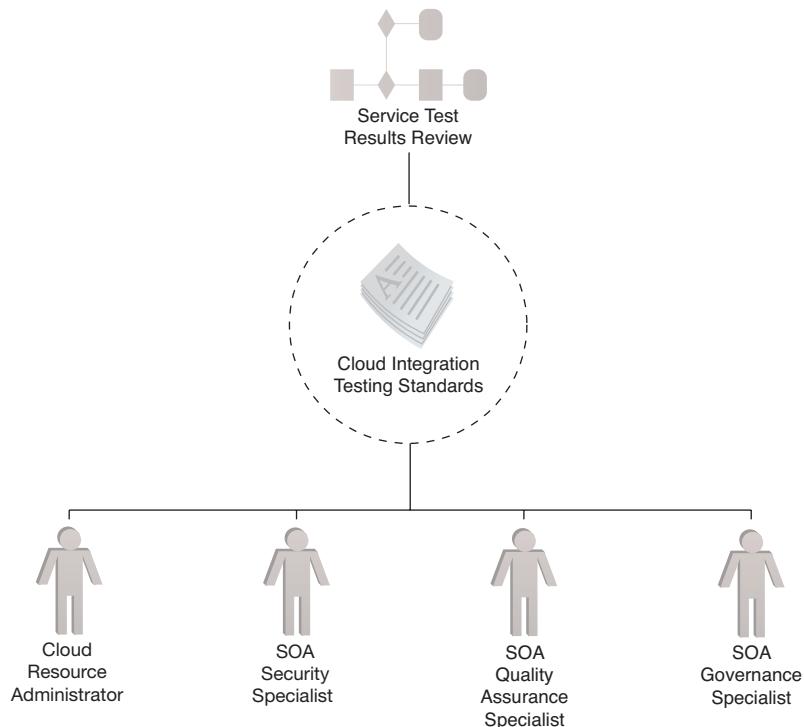


Figure 10.4

The Cloud Integration Testing Standards precept.

Related Roles

- Cloud Resource Administrator
- SOA Security Specialist
- SOA Quality Assurance Specialist
- SOA Governance Specialist
- Other: Cloud Architect
- Other: Cloud Security Specialist
- Other: Cloud Governance Specialist

Test Data Usage Guidelines

Because the testing phase results in the creation of data specific to services and service compositions (such as performance statistics, measured exceptions, etc.), it is important to have guidelines (or, in some cases, even standards) in place to govern the usage of that data.

Some common considerations include:

- Where will test data be stored?
- Who is responsible for the analysis of test data?
- How are testing-related metrics communicated?
- How are testing-related metrics mapped to business requirements?
- How are testing-related metrics mapped to design specifications?
- Who is responsible for the management of test scripts and the configuration of testing tools?

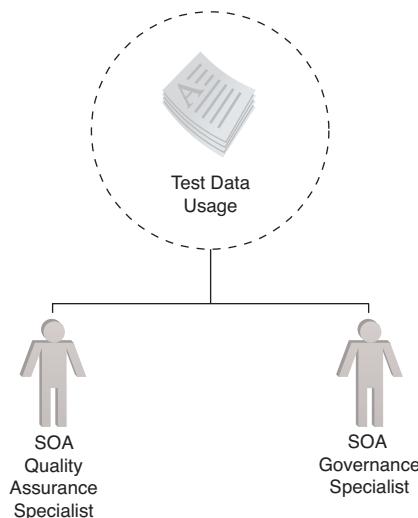


Figure 10.5

The Test Data Usage Guidelines precept.

These and other issues should be addressed by a guidelines document that depicts the flow of test data from its initial raw, collected state to its presentation in metrics reports.

Related Processes

N/A

Related Roles

- SOA Quality Assurance Specialist
- SOA Governance Specialist

Processes

Service Test Results Review

One or more review processes can be established to approve or reject a given service based on the statistics and results made available from whatever tests the service underwent. As part of this review, it should also be confirmed that a given service was, in fact, subjected to all of the appropriate tests.

It is important to note that if failures are encountered in any of the testing steps, the service logic may need to be revised which, in turn, will necessitate cycles of re-testing. Further, test results data can be stored in the service profile as well as a central location (preferably a repository) along with all other service metadata.

Related Precepts

- Testing Parameter Standards
- Service Testing Standards
- Cloud Integration Standards

Related Roles

- SOA Security Specialist
- SOA Quality Assurance Specialist
- SOA Governance Specialist

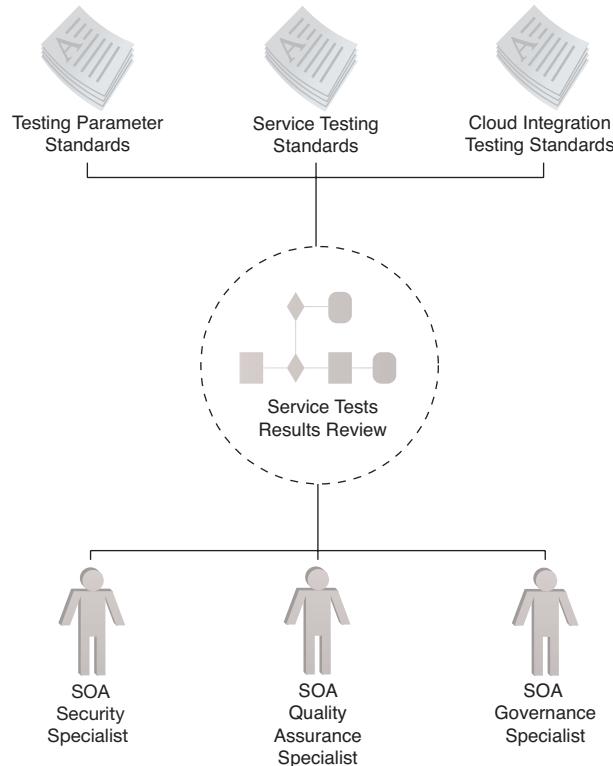


Figure 10.6
The Service Test Results Review process.

People (Roles)

Service Administrator

Service Administrators are usually peripherally involved with establishing governance controls for the Service Testing stage in that they can provide guidance and advice regarding the recommended testing parameters and standards, and the synchronization of testing environments with target production environments.

Related Precepts

- Testing Parameter Standards
- Service Testing Standards

Related Processes

N/A

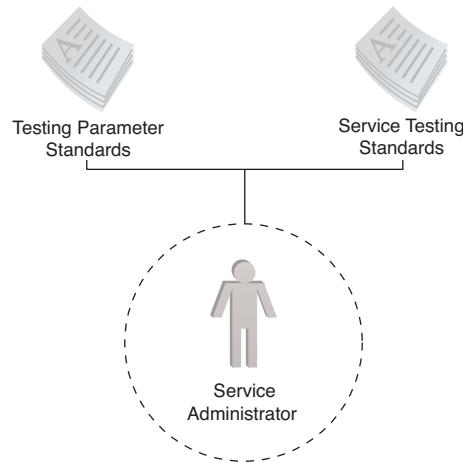


Figure 10.7

Service Testing governance precepts and processes associated with the Service Administrator role.

Cloud Resource Administrator

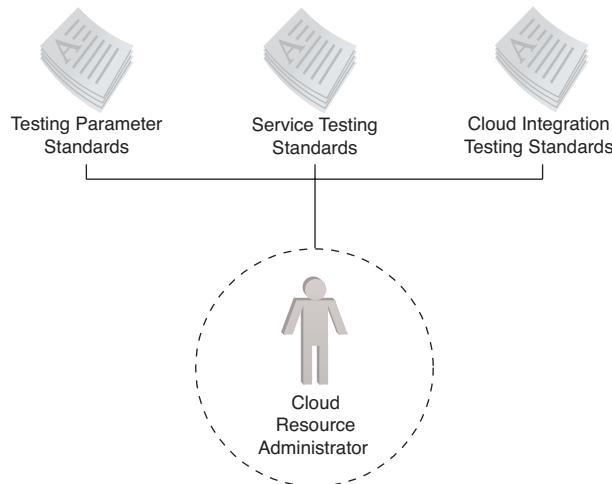
Similar to the role of the Service Administrator, Cloud Resource Administrators provide their expertise and understanding of the target cloud environment for a service to ensure that testing standards are authored correctly and that all cloud-specific considerations are accounted for. When working with PaaS platforms that provide a ready-made environment within which services can be built, tested, and then deployed to production, the Cloud Resource Administer may need to collaborate with the SOA Quality Assurance Specialist to properly govern the Service Testing stage.

Related Precepts

- Testing Parameter Standards
- Service Testing Standards
- Cloud Integration Testing Standards

Related Processes

N/A

**Figure 10.8**

Service Testing governance precepts and processes associated with the Cloud Resource Administrator role.

Enterprise Architect

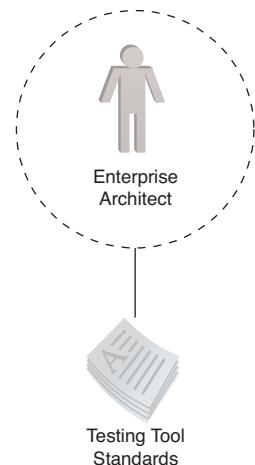
In relation to SOA governance, the primary area of interest for the Enterprise Architect during the Service Testing stage is the toolset used by SOA Quality Assurance Specialists. When different project teams are delivering services for the same service inventory, the standardization of testing tools helps maintain consistency when carrying out any of the precepts associated with this stage.

Related Precepts

- Testing Tool Standards

Related Processes

N/A

**Figure 10.9**

Service Testing governance precepts and processes associated with the Enterprise Architect role.

SOA Quality Assurance Specialist

As the subject matter expert for this stage and also as the role most involved with the actual execution of Service Testing-related project tasks, the SOA Quality Assurance Specialist can provide input to help define appropriate precepts and to further participate as a peer reviewer during the Service Test Results Review process.

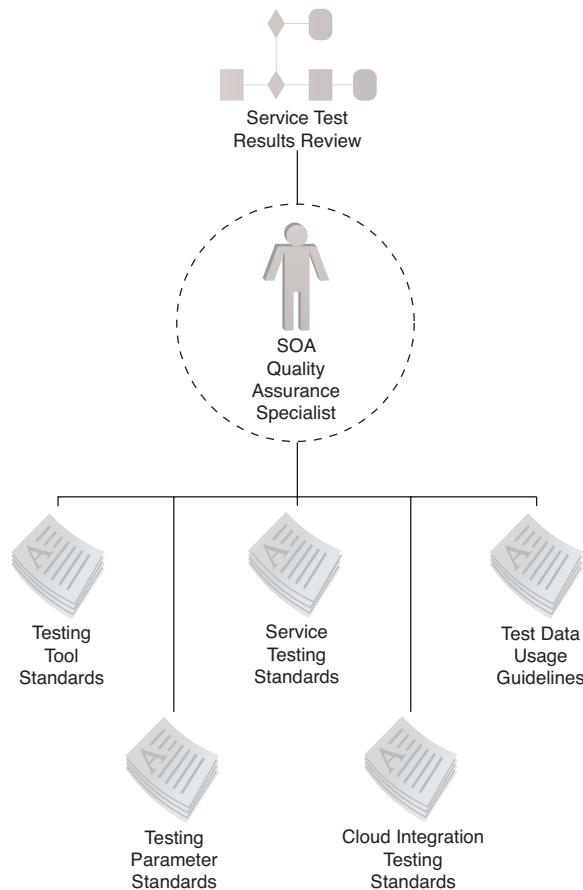


Figure 10.10

Service Testing governance precepts and processes associated with the SOA Quality Assurance Specialist role.

Related Precepts

- Testing Tool Standards
- Testing Parameter Standards
- Service Testing Standards
- Cloud Integration Testing Standards
- Test Data Usage Guidelines

Related Processes

- Service Test Results Review

SOA Security Specialist

As with other project stages, the involvement of the SOA Security Specialist is required when certain security concerns are being addressed within the service architecture. This role will possess the expertise to not only identify types of security-related tests, but will further be able to set the individual thresholds and tolerances pertaining to service security requirements.

For example, the SOA Security Specialist may devise a series of tests to simulate a malicious attack in order to expose encrypted data transmitted in messages sent by a service. The attack may succeed at some point, indicating a certain level of encryption weakness or strength. This level may or may not be deemed acceptable, depending on the standards set by the SOA Security Specialist.

Further, SOA Security Specialists will be particularly involved with testing security controls with cloud-based services, as some of the security mechanisms in use may be supplied by the cloud provider. As briefly explained in Chapter 7, cloud environments can introduce new security concerns, especially in third-party clouds where resources used by the service are shared by other cloud consumer organizations (leading to the need to share corresponding trust boundaries).

Related Precepts

- Testing Parameter Standards
- Service Testing Standards
- Cloud Integration Testing Standards

Related Processes

- Service Test Results Review

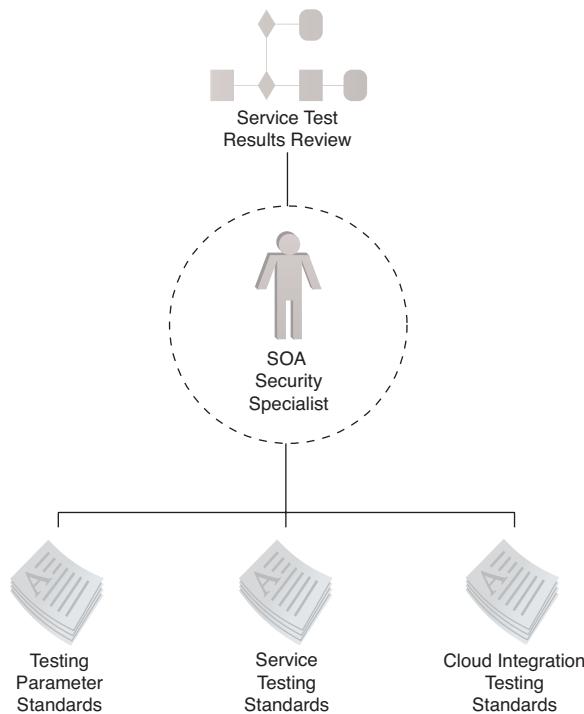


Figure 10.11

Service Testing governance precepts and processes associated with the SOA Security Specialist role.

SOA Governance Specialist

The SOA Governance Specialist participates in an advisory role throughout the definition and application of Service Testing precepts. However, this role can further aid this governing stage by identifying and involving other technology or subject matter experts that may be required to assist with the identification and creation of testing standards, as well as participation during the Service Test Results Review.

Related Precepts

- Testing Tool Standards
- Testing Parameter Standards
- Service Testing Standards

- Cloud Integration Testing Standards
- Test Data Usage Guidelines

Related Processes

- Service Test Results Review

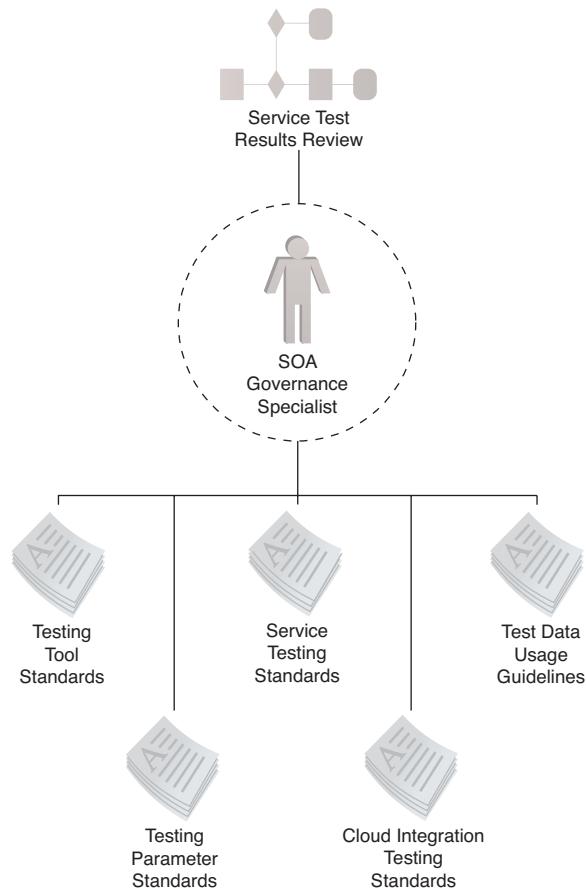


Figure 10.12
Service Testing governance precepts
and processes associated with the SOA
Governance Specialist role.

SUMMARY OF KEY POINTS

- Service Testing governance precepts intend to ensure that services fulfill testing requirements within required parameters and in compliance with pre-defined standards.
- Test data is a primary governance concern to ensure its on-going use and value in the form of metrics.
- Cloud-based services introduce special testing requirements, some of which are related to distinct security concerns.

CASE STUDY EXAMPLE

Following the guidance and requirements set forth by the SOA Governance Program Office precepts, the Raysmore quality assurance department establishes an SOA testing program.

Some distinguishing characteristics of the program include the following:

- a standardized set of performance metrics created for evaluating service performance and SLA compliance
- business-centric parameters documented to verify business requirements compliance

Multiple test environments are made available in order to accurately mirror the production environment in all aspects (except for capacity).

Specifically, the following environments are established:

- development environments for unit testing
- systems functional test environments for systems testing
- performance test environments for performance testing
- pre-production environments for certification testing

Each test environment holds smaller-scale copies of production databases to allow for functional and performance testing with real data. The performance test and pre-production environments have network connectivity to all subsidiaries' test

environments in order to test for service inter-system connectivity and network performance under realistic conditions.

Performance and regression testing is automated using a commercial software product that can generate combinations of random input data. This data can be valid and realistic (for functional or performance testing) or deliberately invalid (for testing that systems generate the correct responses during exception conditions). The testing tool can also run scripts to generate pre-determined volumes of requests to assist performance testing.

For larger service compositions, the SOA Governance Program Office specifies a six week “soak” period in the pre-production environment under a simulated traffic load to determine if there are any further bugs, memory leaks, or other types of flaws.

All initial service testing and certification is performed using Raysmore’s test and staging environments, except for services that wrap legacy systems unique to one or more subsidiaries. Any services that are later deployed to the subsidiary’s operations environment are required to be re-certified for that environment.

The SOA Governance Program Office further mandates that the service repository be used to store test results and capture test history. This allows for the identification of recurring issues and helps keep track of which services are certified for which subsidiary’s production environment.

Following one of the Test Data Management Guidelines published by the SOA Governance Program Office, the quality assurance team creates a standardized testing profile, as shown in Table 10.1.

Test Type	Entry Criteria	Exit Criteria	Next Steps
unit test	<ul style="list-style-type: none">code being tested is complete (if test-driven development practices are used, tests are written before any code is produced)	<ul style="list-style-type: none">all unit tests passcode coverage criteria is met	<ul style="list-style-type: none">store results in service repository along with other service metadataproceed to compliance testing

continues

Test Type	Entry Criteria	Exit Criteria	Next Steps
compliance test	<ul style="list-style-type: none"> • code being tested is complete • compliance rules have been documented 	<ul style="list-style-type: none"> • all compliance tests pass 	<ul style="list-style-type: none"> • store results in service repository along with other service metadata • proceed to security testing • deploy code into system test environment
security test	<ul style="list-style-type: none"> • code being tested is complete • security policies have been documented and implemented 	<ul style="list-style-type: none"> • all security tests pass 	<ul style="list-style-type: none"> • store results in service repository along with other service metadata • proceed to functional testing
functional test	<ul style="list-style-type: none"> • code being tested is complete and unit tested • test cases are defined • optional: functional tests are automated 	<ul style="list-style-type: none"> • all functional tests pass 	<ul style="list-style-type: none"> • store results in service repository along with other service metadata • proceed to regression testing
regression test	<ul style="list-style-type: none"> • code being tested is complete and unit and functionally tested • regression test cases have been documented • optional: regression tests are automated 	<ul style="list-style-type: none"> • all regression tests pass 	<ul style="list-style-type: none"> • store results in service repository along with other service metadata • add functional tests to the regression testing suite • proceed to performance testing • deploy code into performance test environment

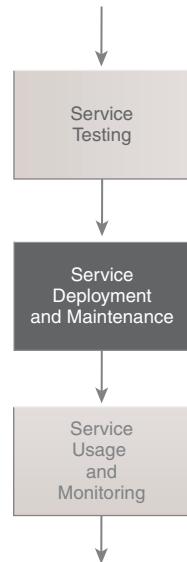
Test Type	Entry Criteria	Exit Criteria	Next Steps
performance test	<ul style="list-style-type: none">• code being tested is complete and unit, functionally, and regression tested• performance test cases are defined for the new consumer• performance test cases are defined for the existing consumers	<ul style="list-style-type: none">• all non-functional criteria is met for the new consumer• combined non-functional criteria set is met for all service consumers• all SLA requirements are met	<ul style="list-style-type: none">• store results in service repository along with other service metadata• add new performance tests to the performance testing suite• deploy code to production

Table 10.1
A testing profile that organizes service testing and related governance steps.

10.2 Governing Service Deployment and Maintenance

Production environments in most IT enterprises are supported by governance controls. With service-oriented solutions these controls can become more stringent. For example, because a single service deployment can establish a point of failure for multiple future service compositions, the amount of checkpoints and rigor that deployments for a reusable service must undergo can be significant.

The primary goal of governing the deployment of services within a production environment is to ensure the stability of the services and to further guarantee that all maintenance changes face the appropriate scrutiny to preserve the service's (stable) production status.



NOTE

It is important, for the purpose of this chapter, to distinguish service maintenance from service versioning. This governance stage is concerned with maintenance issues that usually relate to an initial service deployment. Technology upgrades and bug fixes are the two most common types of maintenance deployments that fall under this category.

The upcoming *Governing Service Versioning and Retirement* section in Chapter 11 is dedicated solely to service versioning issues. Depending on the configuration system and versioning strategy being employed, any maintenance deployment may actually constitute a new service version (in which case you should consider the maintenance-related content in this chapter part of the Service Versioning and Retirement stage).

Precepts

Production Deployment and Maintenance Standards

A typical IT enterprise will have already established governance requirements for any software programs to be deployed or upgraded in the production environment.

These requirements can include:

- production entrance criteria
- specific steps to be followed for production deployments and upgrades
- forms and approvals needed to apply for and carry out a deployment or an upgrade
- guidelines for dealing with common production deployments issues

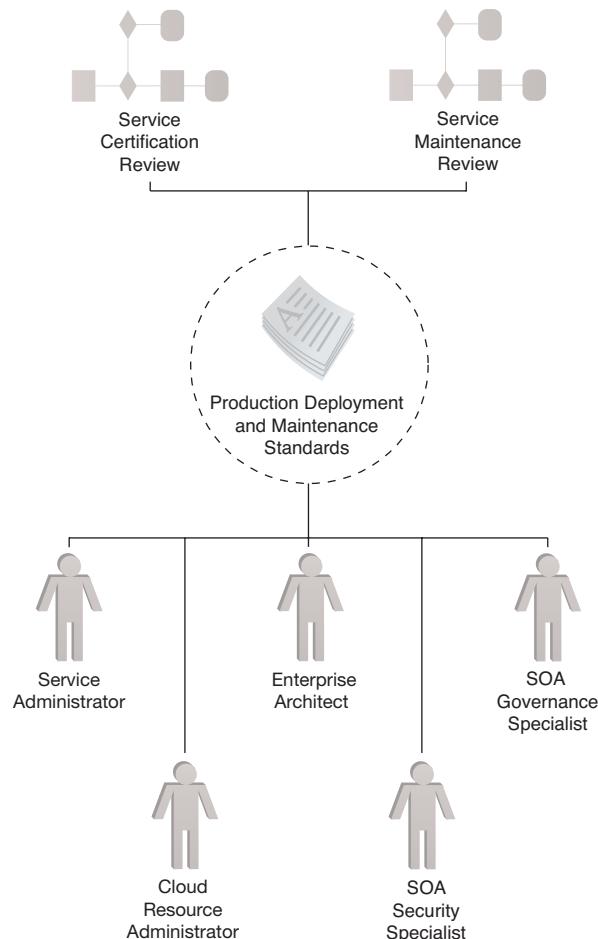


Figure 10.13

The Production Deployment and Maintenance Standards precept.

The deployment and maintenance of services can introduce additional considerations, especially in relation to surrounding services that share the same service inventory environment. For example, it may be necessary to analyze the impact of a newly deployed, reusable service on pooled infrastructure resources that are already being shared by other services.

Upgrades and bug fixes can be challenging because they carry a potential impact to the stability of the existing service and its consumers.

Example considerations include:

- Is enough data being captured to resolve bugs quickly and efficiently?
- Is the resolution process followed to introduce a bug fix consistent with issue severity?
- For a cloud-based service deployment, are there required procedures specified by the cloud provider that must be followed?

This precept represents the deployment and maintenance requirements, standards, and guidelines specific to the target production environment for a given service. As noted in the last bullet from the preceding list, services deployed in public cloud environments may be subject to deployment and maintenance limitations imposed by third-party cloud providers. In this case, the resulting standards and guidelines may need to be based on or defined in compliance with these limitations.

Related Processes

- Service Certification Review
- Service Maintenance Review

Related Roles

- Service Administrator
- Cloud Resource Administrator
- Enterprise Architect
- SOA Security Specialist
- SOA Governance Specialist

Processes

Service Certification Review

Before any new service is deployed into a production environment, it must receive formal approval. Typically, this is achieved as a result of the completion of the Service Certification Review process, which is necessary to ensure that:

- the correct service is being deployed to the correct location
- the deployment location is consistent with the preceding testing and staging environments (a consideration especially relevant when deploying services to third-party cloud platforms)
- all advertised quality characteristics (as documented in the SLA and service profile) have been fulfilled
- any required service monitoring and management tools and processes are in place
- all applicable billing rates (for services deployed on leased IT resources) have been verified

Many individuals can participate in this process, each potentially involved in a different aspect of the review. If, subsequently, the review is successful, the service is considered “certified” and ready to enter the production environment. Prior to this point, production support teams should have been notified so that they are prepared for the new service deployment and its requirements. (This type of notification can contain a pointer to the corresponding service registry record.)

Related Precepts

- Production Deployment and Maintenance Standards

Related Roles

- Service Administrator
- Cloud Resource Administrator
- Enterprise Architect
- Service Custodian
- SOA Quality Assurance Specialist
- SOA Security Specialist

- SOA Governance Specialist
- Other: Cloud Architect
- Other: Cloud Security Specialist
- Other: Cloud Governance Specialist

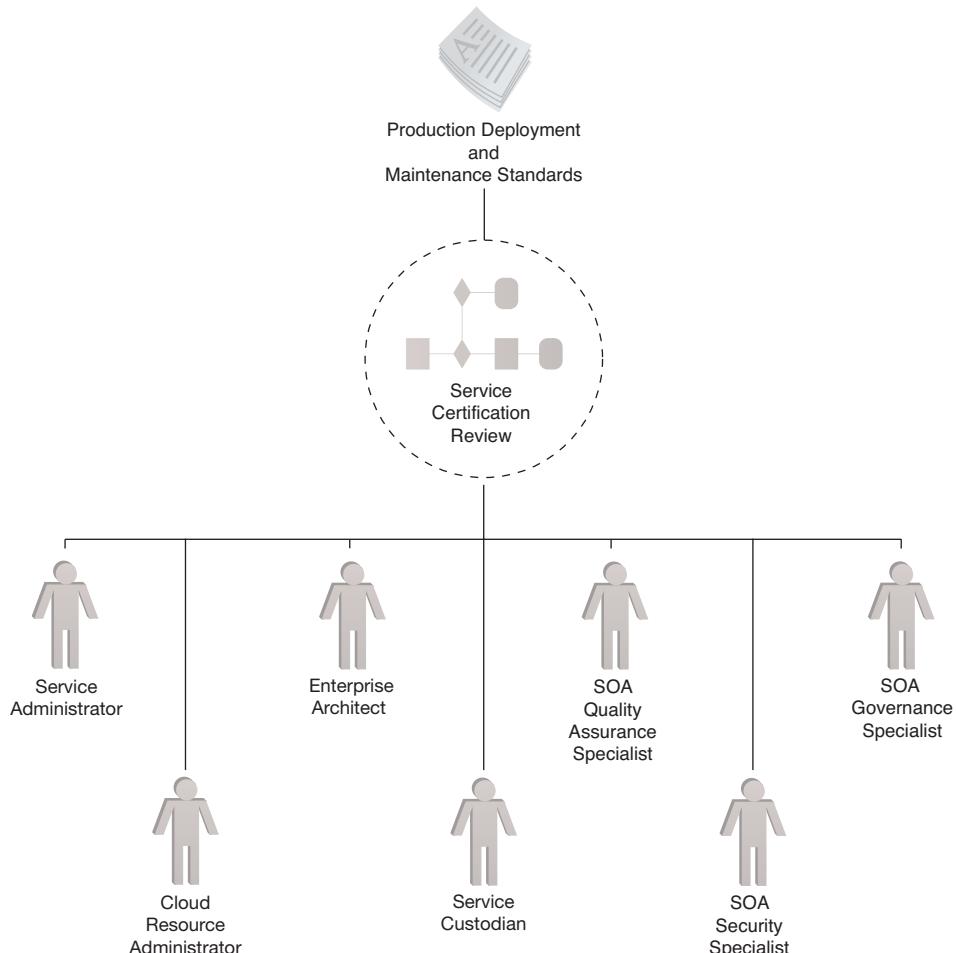


Figure 10.14

The Service Certification Review precept.

Service Maintenance Review

Maintenance changes that affect the service code, such as bug fixes, must be stored in the source code repository alongside the original service code. These changes need to be introduced in such a way that they do not interfere with any new work being performed against the same code base. In most typical development platforms this is achieved by introducing a new branch in the source control tool. Although several roles can be involved in this process, the Service Custodian in particular needs to approve any such changes prior to deployment. Maintenance changes can be further logged in the service's registry record.

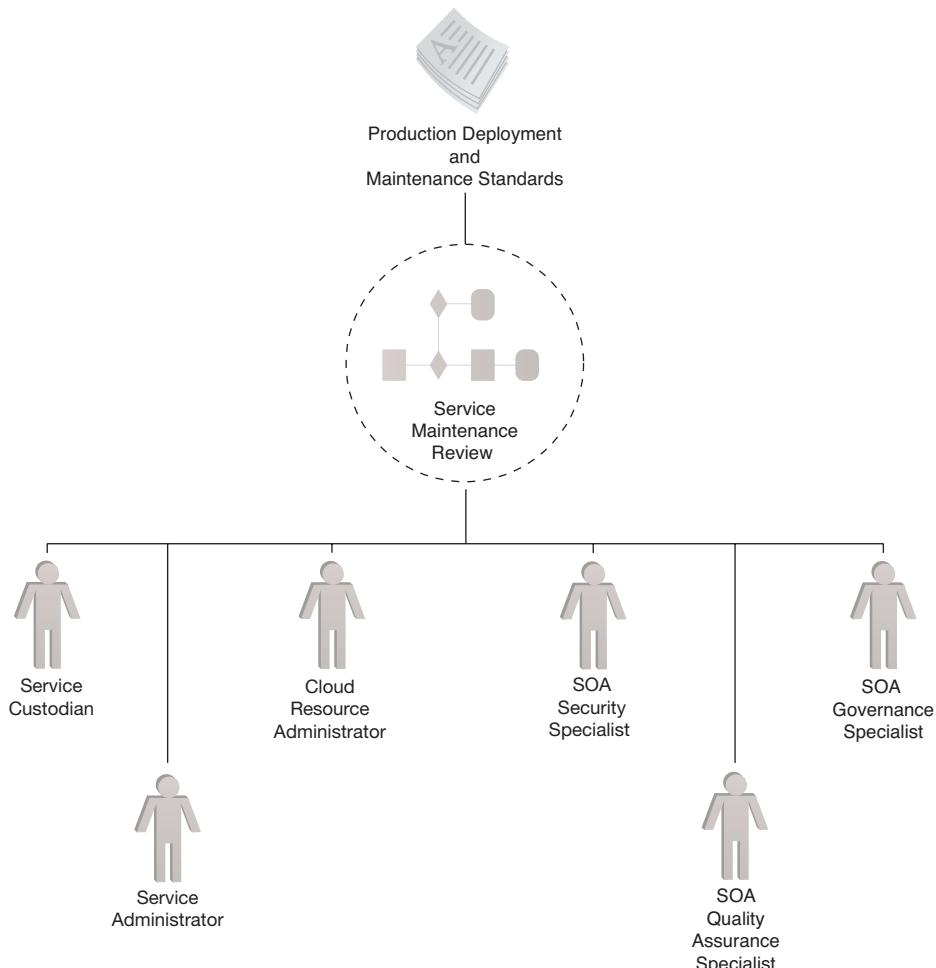


Figure 10.15

The Service Maintenance Review precept.

If maintenance changes are classified as fixes or upgrades that do not constitute a new major service version, then, depending on the change management system and methodology in use by the IT department, it may or may not be necessary to subject the revised service to another complete Service Certification Review.

NOTE

In some cases, the Production Deployment and Maintenance Requirements precept will dictate that if a maintenance change warrants a re-certification of the service, it must result in a new service version and therefore become subject to further service versioning precepts.

Related Precepts

- Production Deployment and Maintenance Standards

Related Roles

- Service Custodian
- Service Administrator
- Cloud Resource Administrator
- SOA Quality Assurance Specialist
- SOA Security Specialist
- SOA Governance Specialist
- Other: Cloud Computing Security Specialist
- Other: Cloud Computing Governance Specialist

People (Roles)*Service Administrator*

For on-premise service implementations, the Service Deployment and Maintenance stage is primarily associated with this role. The Service Administrator, together with the Service Custodian and SOA Governance Specialist, typically lead the efforts to establish the Production Deployment and Maintenance Standard precept and associated processes.

Related Precepts

- Production Deployment and Maintenance Standards

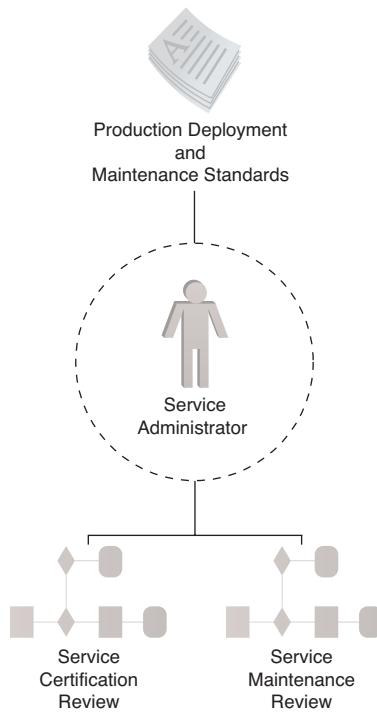


Figure 10.16

Service Deployment governance precepts and processes associated with the Service Administrator role.

Related Processes

- Service Certification Review
- Service Maintenance Review

Cloud Resource Administrator

For services destined for deployment within a cloud environment, this role will lead necessary governance activities together with the Service Custodian and SOA Governance Specialist. The Cloud Resource Administrator's involvement may not be limited to the service implementation, as further cloud-based IT resources may need to be set up and maintained to ensure consistent performance behavior, especially in cloud environments shared by multiple cloud consumer organizations.

A primary task during this stage is the configuration of on-demand and dynamic scaling settings. Mechanisms, such as the automated scaling listener and the pay-for-use monitor, may need to be positioned and tuned via cloud-based desktop tools so that the cloud service can leverage cloud provided IT resources within pre-defined parameters.

Related Precepts

- Production Deployment and Maintenance Standards

Related Processes

- Service Certification Review
- Service Maintenance Review

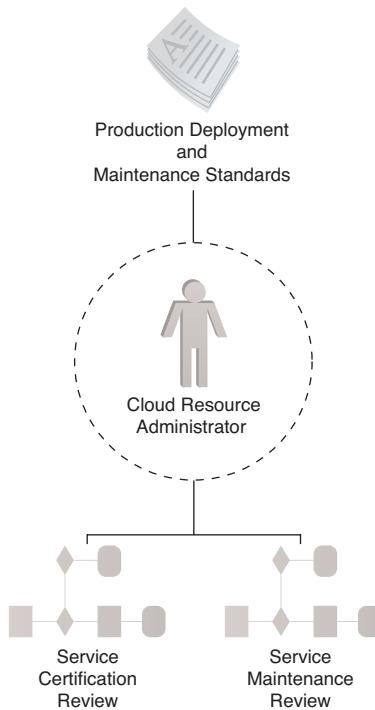


Figure 10.17

Service Deployment governance precepts and processes associated with the Cloud Resource Administrator role.

Service Custodian

Although the owner or custodian of a service may have been involved in previous service lifecycle stages, it is often as part of Service Deployment where ownership is officially assigned to a designated Service Custodian. As a result, this role will be involved with the Service Certification Review, during which it will act as a primary point of contact for any issues or objections that may arise. The Service Custodian is also a standard participant of Service Maintenance Reviews, as this individual may be required to approve any planned upgrades or fixes that affect the service architecture.

Related Precepts

N/A

Related Processes

- Service Certification Review
- Service Maintenance Review

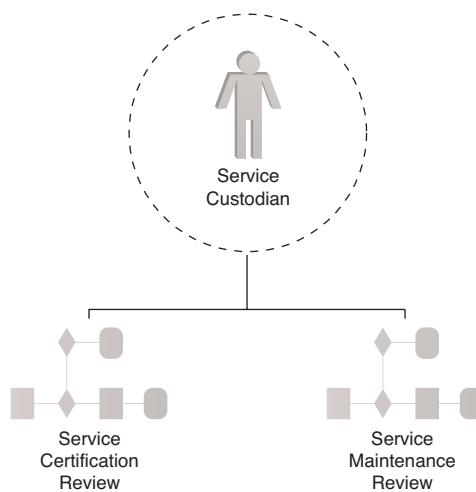


Figure 10.18

Service Deployment governance precepts and processes associated with the Service Custodian role.

Enterprise Architect

Many of the standards and requirements defined in the Production Deployment and Maintenance Standards specifications will be influenced by or even derived from underlying platform technology architecture and infrastructure. Enterprise Architects therefore can become a primary contributor to these standards and their sign-off may further be required for the deployment of any service implementation that introduces new or previously non-standardized technologies or resources.

Related Precepts

- Production Deployment and Maintenance Standards

Related Processes

- Service Certification Review

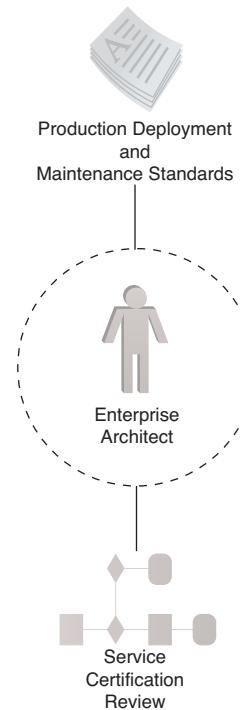


Figure 10.19

Service Deployment governance precepts and processes associated with the Enterprise Architect role.

SOA Quality Assurance Specialist

Quality assurance concerns can extend to the Service Deployment stage in order to validate that a service is functioning and performing in the production environment as it was previously when in testing and staging environments. In this capacity, the SOA Quality Assurance Specialist can take part in the Service Certification Review and Service Maintenance Review processes.

Related Precepts

N/A

Related Processes

- Service Certification Review
- Service Maintenance Review

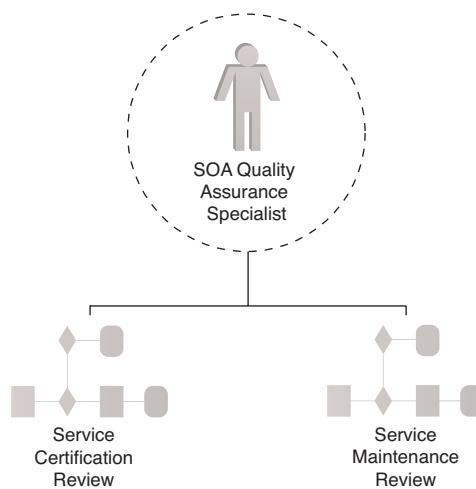


Figure 10.20

Service Deployment governance precepts and processes associated with the SOA Quality Assurance Specialist role.

SOA Security Specialist

The SOA Security Specialist will be asked to join the Service Certification Review and/or the Service Maintenance Review process when it's necessary to perform a security audit on the service's production implementation. This may involve assessing the service as it relates to and interacts with other services.

The Production Deployment and Maintenance Standards may also benefit from having the SOA Security Specialist contribute to ensure that existing standards don't inhibit the use of required security mechanisms or inadvertently introduce security vulnerabilities.

Related Precepts

- Production Deployment and Maintenance Standards

Related Processes

- Service Certification Review
- Service Maintenance Review

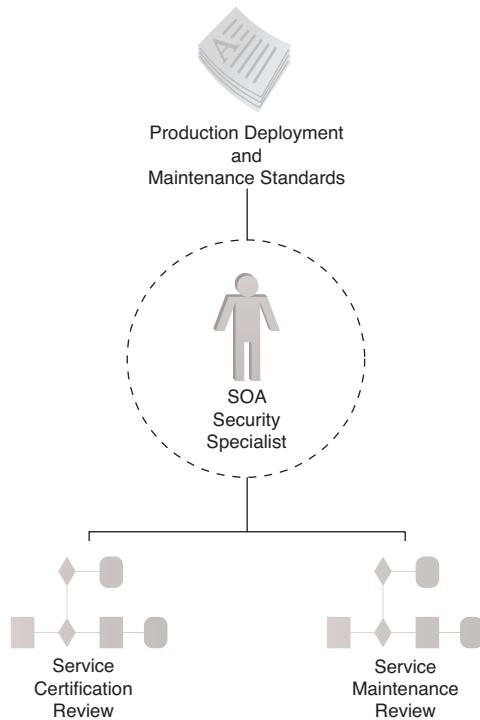


Figure 10.21

Service Deployment governance precepts and processes associated with the SOA Security Specialist role.

SOA Governance Specialist

In addition to providing guidance and coordination for the transition of the developed and tested service to its production deployment, the SOA Governance Specialist can further help ensure that Service Deployment exit criteria is in alignment with Service Usage precepts that will already be in place.

Related Precepts

- Production Deployment and Maintenance Standards

Related Processes

- Service Certification Review
- Service Maintenance Review

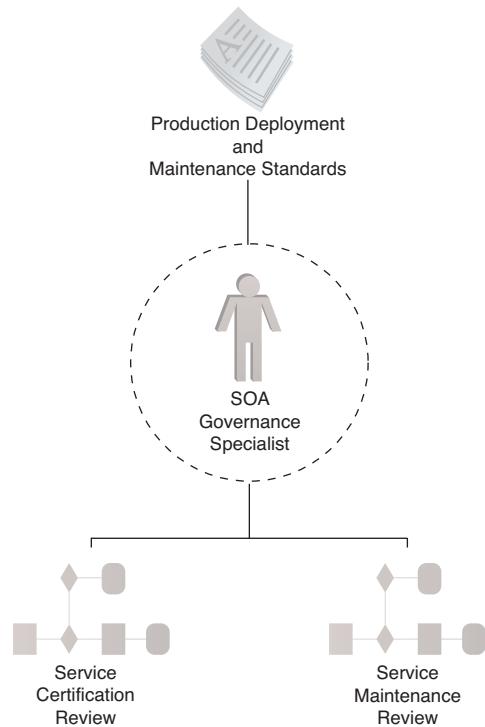


Figure 10.22

Service Deployment governance precepts and processes associated with the SOA Governance Specialist role.

NOTE

The Schema Custodian and Policy Custodian may be asked to participate in the Service Deployment stage to ensure that implementation of schemas and policies (especially when shared by multiple services) is carried out correctly. Often their involvement is required to resolve conflicts that may occur with runtime schema and policy processing and validation. However, these roles are not usually required to contribute to governance tasks.

Of course, other IT professionals, such as System Administrators (and various production support personnel), are also commonly involved in the actual management and execution of deployment and maintenance processes.

SUMMARY OF KEY POINTS

- Changes to the production environment needs to be introduced in a very deliberate and pragmatic fashion, especially when dealing with shared services.
 - Before any new service code is deployed into the production environment, it must receive formal approval, typically through a Service Certification Review.
 - Maintenance deployments and upgrades can constitute a new major service version and therefore can be subject to additional governance controls.
-

CASE STUDY EXAMPLE

The SOA Governance Program Office recognized early on that Raysmoore required a rapid and effective resolution of maintenance issues (such as minor software bugs and unforeseen SLA violations) pertaining to newly deployed services. Rapid problem resolution and consistently meeting promised service SLAs was a critical success factor clearly defined in the original SOA roadmap and one that is becoming increasingly important as the new Raysmoore service inventory grows steadily more reliant on agnostic services to support core business operations.

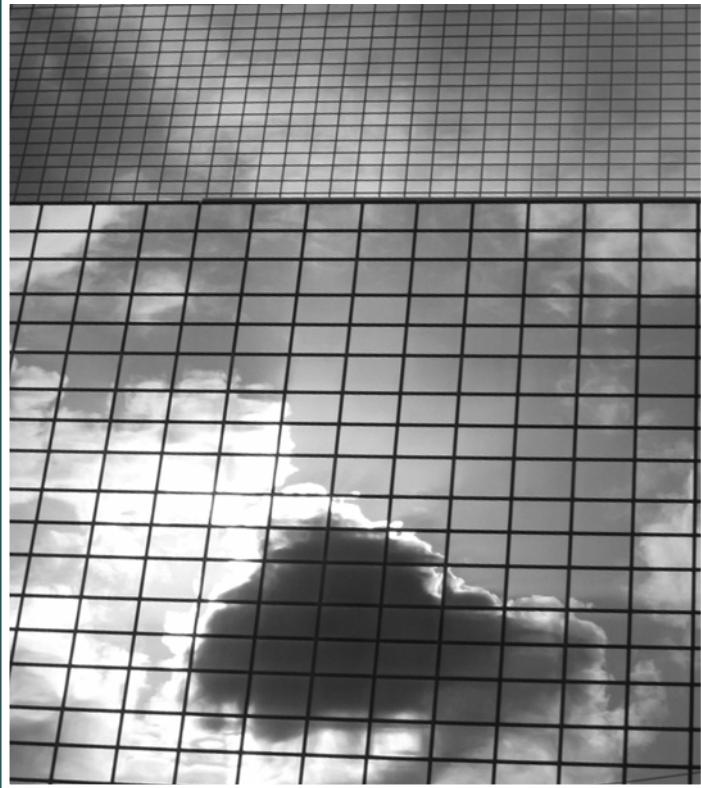
To this end, the SOA Governance Program Office arranges for the help desk to handle technical problems and SLA violations with services, according to pre-defined severity codes:

- *Severity 1 (service unavailable)* – Respond to consumers within one hour during normal business hours, or within three hours during evenings and weekends. Target 95% resolution of Severity 1 problems in the same business day.
- *Severity 2 (significant impairment of service function or violation of SLA by 20% or more)* – Respond to consumer within two hours during normal business hours or next business day. Target of 95% resolution of Severity 2 problems within two working days.
- *Severity 3 (minor functional issues or SLA violations)* – Respond to consumer within two working days. Target 95% of Severity 3 problems to be resolved within five working days.

When a problem occurs, the fix is made available to all service consumers as soon as the service has completed any necessary regression testing and re-certification. In order to meet the deadlines for Severity 1 and 2 issues, emergency certification reviews can be called at any time.

The SOA Governance Program Office helps establish a process that ensures that all the changes to the code made as a result of bug fixes are checked into a central code repository. (This helps avoid a previous problem where support personnel inadvertently re-introduced previously identified bugs into new service deployments.)

This page intentionally left blank



Chapter 11

Governing Service Usage, Discovery, and Versioning Stages

11.1 Governing Service Usage and Monitoring

11.2 Governing Service Discovery

11.3 Governing Service Versioning and Retirement

SOA PRINCIPLES & PATTERNS REFERENCED IN THIS CHAPTER

- Compatible Change [505]
- Cross-Domain Utility Layer [511]
- Metadata Centralization [536]
- Proxy Capability [547]
- Service Abstraction (478)
- Service Decomposition [556]
- Service Discoverability (484)
- Service Normalization [563]
- Service Refactoring [565]
- Termination Notification [569]
- Version Identification [575]

The SOA governance precepts covered in this chapter are dedicated to post-deployment stages during which services are active and available for use by service consumers, and further subject to evolutionary changes.

11.1 Governing Service Usage and Monitoring

The runtime governance of deployed services is a critical regulatory focal point because it represents the stage during a service's lifecycle during which we have the opportunity to actively receive value and benefit in return for the investment we made to create the service.

Precepts for service usage governance are, for the most part, dedicated to establishing parameters that limit how services can be used. These precepts are geared to protecting the service from excess or inappropriate usage so as to ensure stability, reliability, and consistent behavior. This, in return, protects all service consumers that rely on the service. Furthermore, these parameters help ensure that the service does not compromise resources shared by other services within the same service inventory or shared by other parts of the IT enterprise.

The Service Usage and Monitoring stage is also the part of the service lifecycle during which the majority of runtime metrics are collected both for use by precepts in processes related to this stage, but also for use by other (pre- and post-deployment) precepts and processes, to help assess their effectiveness.

Precepts

Runtime Service Usage Thresholds

This precept establishes a set of thresholds, some applied generally to most or all services within a service inventory, others are applied or adjusted specifically for individual services.

Common types of usage thresholds include:

- *Service Composition Membership Threshold* – The amount of service compositions an agnostic service can participate in. This value is usually based on scalability limitations or limitations imposed by shared and legacy resources encapsulated by the service.
- *Service Instance Threshold* – This value represents the amount of instances of a service that can exist concurrently. This scalability limitation is usually based on available infrastructure and memory resources.

- *Cloud Burst Threshold* – This threshold represents the point at which a given service will scale into a cloud. Often, the Cloud Burst Threshold is the same as the Service Instance Threshold in that the latter value represents the on-premise limit which, when reached, prompts further service instances to be invoked in a correlated cloud-hosted environment.
- *Service Billing Threshold* – When pay-per-usage mechanisms are used to monitor service usage, this value can represent a limit imposed by the usage budget (or credit) associated with the service configuration. This type of threshold is common in cloud-based environments where services are deployed on leased infrastructure resources that support dynamic and on-demand scaling.
- *Service Elasticity Threshold* – Service elasticity is a general measure of a service's dynamic scalability. This type of threshold is broader than other scalability-related thresholds and may therefore be used as a baseline or as a parent limitation for other thresholds (such as those pertaining to service billing or service instances).
- *Service Exception Threshold* – Services runtime exceptions can be the result of problems originating from inside the service architecture or from its surrounding infrastructure or even from other services participating in the same runtime activity. When a service encounters an abnormally large amount of exceptions, it can be an indication that the service (or a part of its surroundings) is being attacked or that there are critical issues with its implementation. Either way, when this threshold is reached, it is often necessary to shut the service down and, if available, invoke a failover system to defer processing to a backup service implementation.
- *Service Data Throughput Threshold* – Various factors can be taken into account to set the data throughput limit of a given service. For example, the data may be subject to complex calculations that consume memory and therefore require the quantity of data to be limited. Or, there may be bandwidth or connectivity limitations, especially when transmitting data to remote or geographically disbursed cloud-based services via third-party internet connections.
- *Service Monitoring Footprint Threshold* – There are several governance products and platforms that introduce sophisticated monitoring features, usually implemented via intelligent, processing-rich service agents. These types of event-driven programs can track individual services and service instances, collecting data for

metrics and invoking notification and logging routines, as required. In larger environments with greater quantities of services and service compositions residing alongside each other, the amount of runtime processing and memory consumed by monitoring-related programs can take its toll on the overall resources shared by services. Therefore, governance thresholds that limit service monitoring may be required.

An important consideration when applying this precept is that some thresholds may need to be set at the service capability level instead of at the service level. For example, different service capabilities within the same service may be assigned different Service Billing Thresholds depending on the nature and importance of their processing logic.

There are many more thresholds that can be established by this precept. Some may be specific to the monitoring product or technology being used, while others may be specific to business requirements or general project constraints.

Related Processes

- Service Vitality Review

Related Roles

- Service Administrator
- Cloud Resource Administrator
- Enterprise Architect
- Service Architect
- Service Custodian
- SOA Security Specialist
- SOA Governance Specialist
- Other: Cloud Architect
- Other: Cloud Security Specialist
- Other: Cloud Governance Specialist

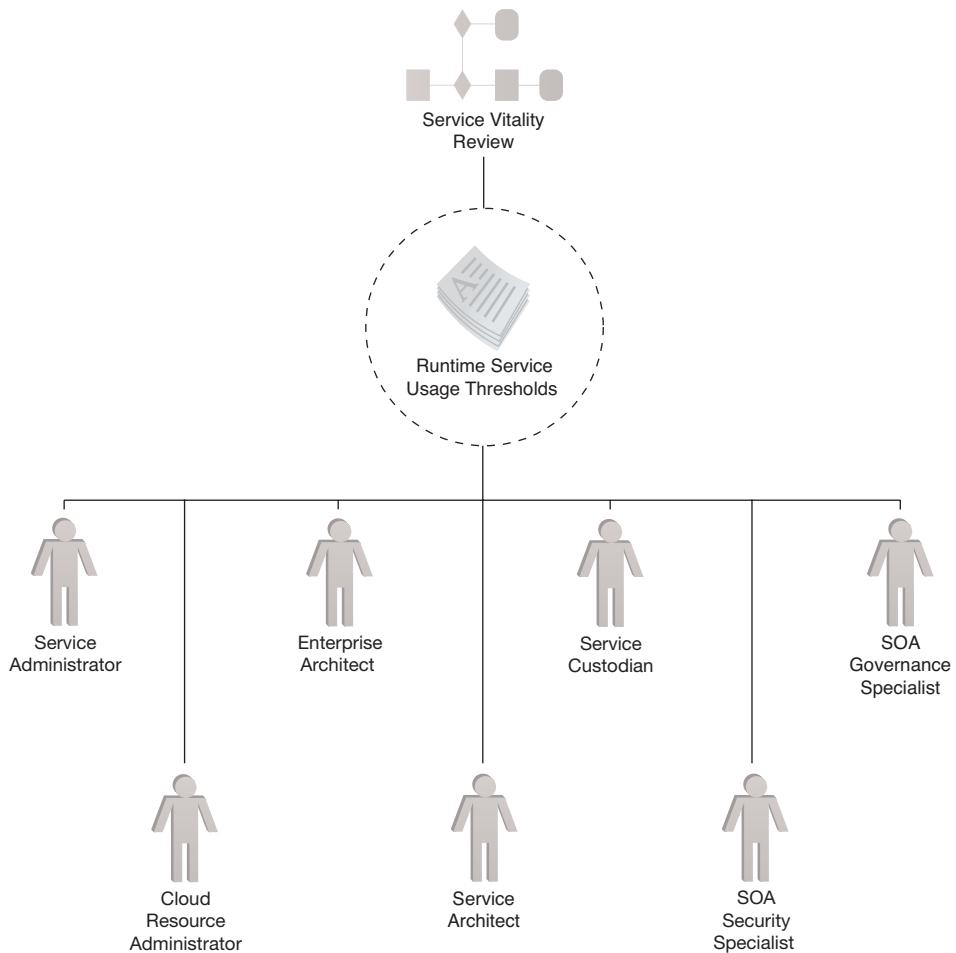


Figure 11.1
The Runtime Service Usage Thresholds precept.

Service Vitality Triggers

Over time, after a service is in production and actively being used by service consumers, various events can impact the service logic, its implementation, or resources it may depend upon. In order to maintain maximum value as an IT asset, it is generally necessary to review and check on the service's "vitality" to either confirm that it is still in an optimum state or to confirm that it requires attention.

Service vitality triggers represent anticipated events that execute vitality activities as part of a Vitality Review process. These activities step those involved with the service's governance through a series of considerations to determine whether or not the service should be subjected to a "refresh" in order to update or adjust any part of its implementation.

Common vitality triggers include:

- Strategic Business Adjustment
- Strategic IT Adjustment
- Business Shift
- Technology Shift
- Performance Metrics
- Compliance Metrics
- Scheduled Milestone
- Scheduled Time Period

Note that the Performance Metrics and Compliance Metrics vitality triggers generally encompass metrics collected in relation to the thresholds established by the Runtime Service Usage Thresholds precept. For example, if an allowable threshold is exceeded, it may warrant a violation logged as a compliance metric. If service responsiveness fluctuates, it may be measured in relation to corresponding usage thresholds.

The listed vitality triggers are described individually in Chapter 13.

Related Processes

- Service Vitality Review

Related Roles

- Service Administrator
- Cloud Resource Administrator
- Enterprise Architect
- Service Architect
- Service Custodian

- SOA Security Specialist
- SOA Governance Specialist
- Other: Cloud Architect
- Other: Cloud Security Specialist
- Other: Cloud Governance Specialist

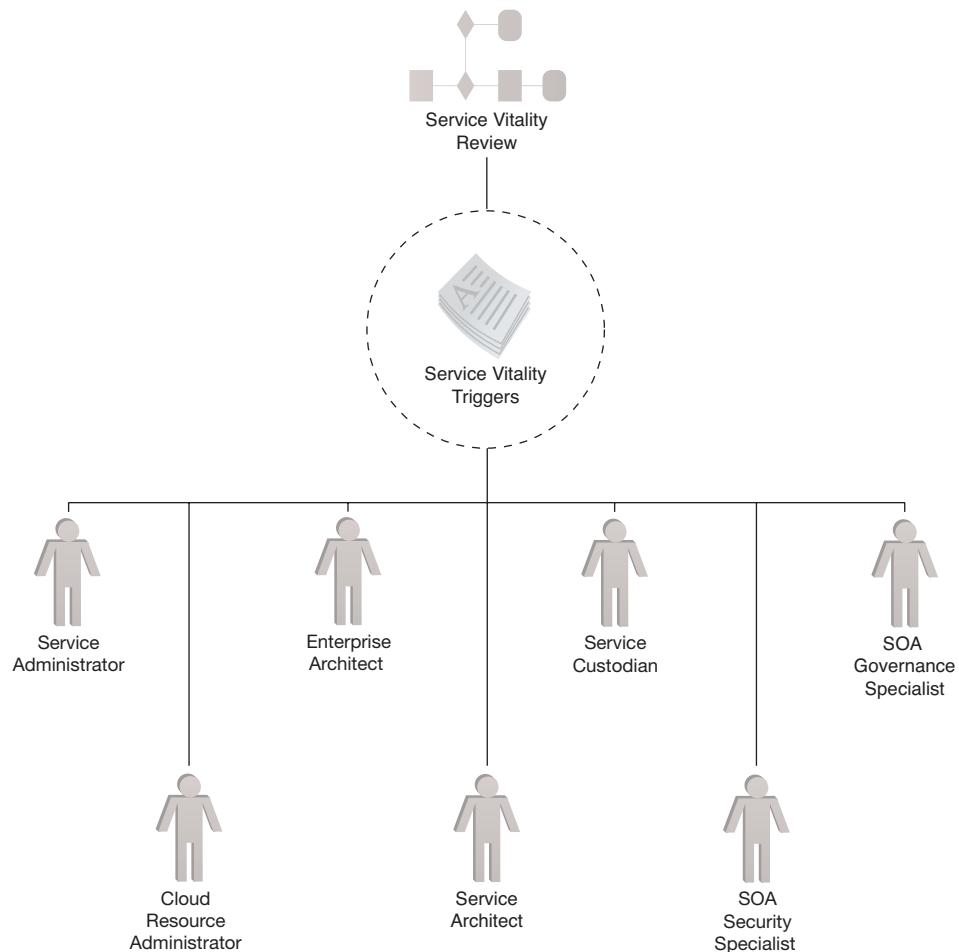


Figure 11.2
The Service Vitality Triggers precept.

Processes

Service Vitality Review

When a vitality trigger is executed, it initiates a review of the service status and implementation. Each such review aims to assess the service vitality in order to determine whether there is a need (and justification) to make improvements and the extent of improvements required.

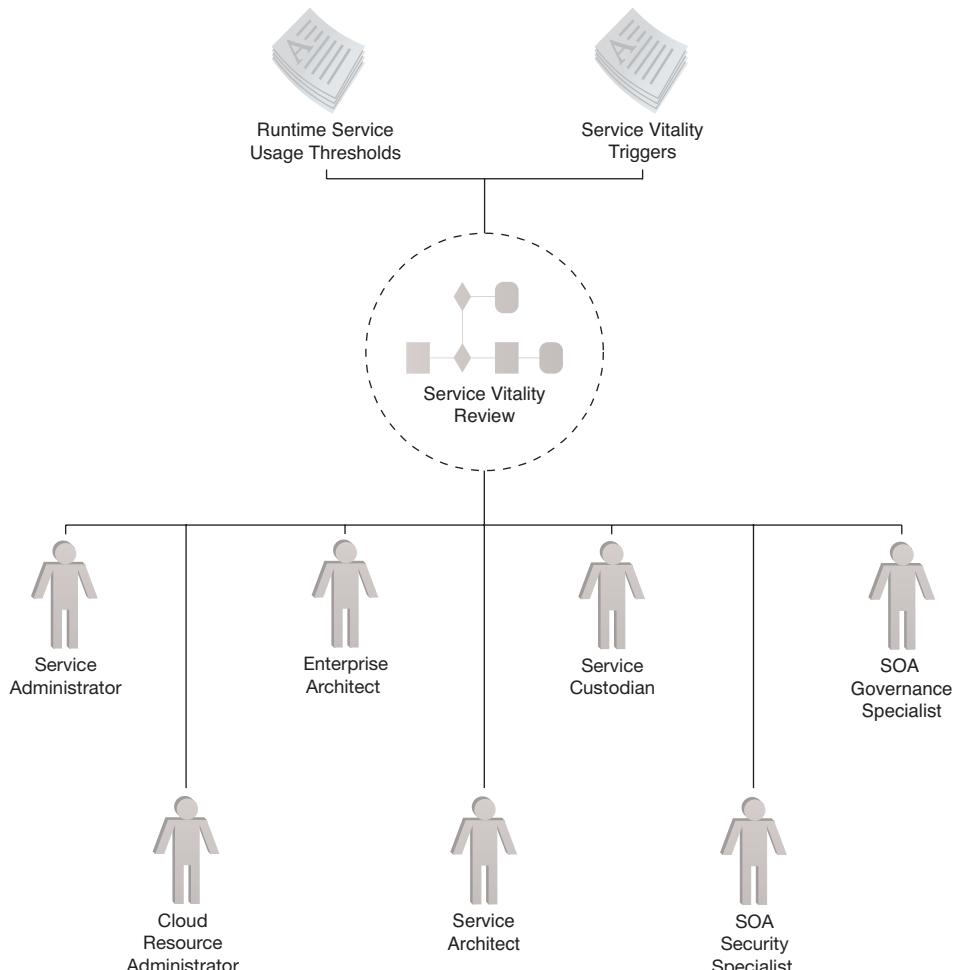


Figure 11.3

The Service Vitality Review process.

For example, the outcome of a service vitality review could result in a recommendation that the service be re-factored, discontinued, superseded by a new version, or that no changes are needed at all.

The base vitality process that is executed to perform this assessment and, if necessary, act upon it, contains the following common steps:

- Identify Activity
- Assess Activity
- Refresh Activity
- Approve Activity
- Communicate Activity

These steps are explained separately in Chapter 13. Custom vitality review processes are generally required based on the organization's preferences, the nature of the vitality trigger responsible for initiating the review, and the nature of metrics collected relevant to the trigger and review. Some metrics will likely correspond to the parameters established by the Runtime Service Usage Thresholds precept, in which case the review can further act as a means of assessing the effectiveness of service usage thresholds.

Related Precepts

- Runtime Service Usage Thresholds
- Service Vitality Triggers

Related Roles

- Service Administrator
- Cloud Resource Administrator
- Enterprise Architect
- Service Architect
- Service Custodian
- SOA Security Specialist
- SOA Governance Specialist
- Other: Cloud Architect
- Other: Cloud Security Specialist
- Other: Cloud Governance Specialist

People (Roles)

Enterprise Architect

Because the service, during the Service Usage and Monitoring stage, resides and actively participates in the production environment of an IT enterprise, the Enterprise Architect will generally be involved in any related precepts or processes. The extent of required participation will often depend on the extent to which a service implementation relies upon or accesses shared resources and legacy systems. For example, service implementations with high levels of autonomy (or those deployed in isolated or cloud-based environments) may require the Enterprise Architect to be only peripherally involved.

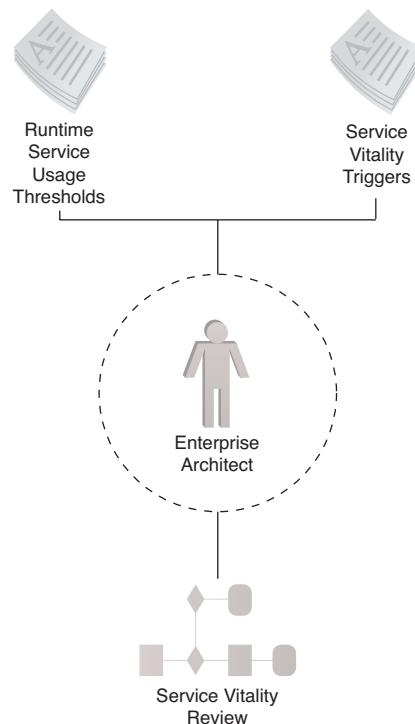


Figure 11.4

Service Usage and Monitoring governance precepts and processes associated with the Enterprise Architect role.

Related Precepts

- Runtime Service Usage Thresholds
- Service Vitality Triggers

Related Processes

- Service Vitality Review

Service Architect

The Service Architect has intimate knowledge of the service implementation and any related resources or mechanisms (such as those that may have been introduced by service composition architectures). Therefore, when setting usage thresholds or exploring vitality improvements, the Service Architect will generally become a primary point of contact and will often end up negotiating (together with the Service Custodian) proposed improvements and refresh updates with the Enterprise Architect.

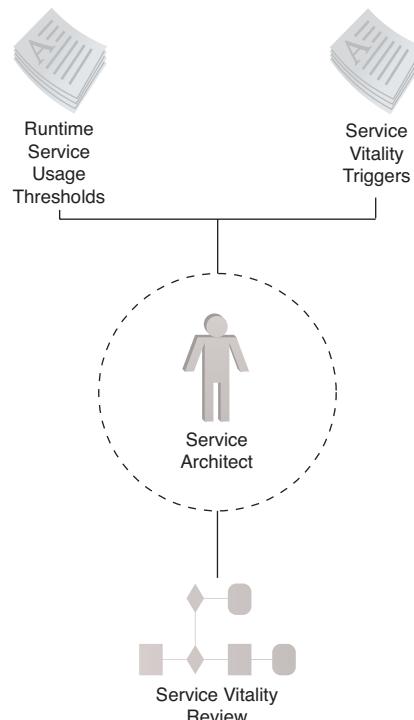


Figure 11.5

Service Usage and Monitoring governance precepts and processes associated with the Service Architect role.

Related Precepts

- Runtime Service Usage Thresholds
- Service Vitality Triggers

Related Processes

- Service Vitality Review

Service Administrator

Throughout a service's runtime existence, the Service Administrator will be a central part of its on-going usage monitoring and performance maintenance. As a result, this role is a primary participant in establishing the necessary usage threshold and vitality trigger governance precepts, as well as associated Service Vitality Reviews.

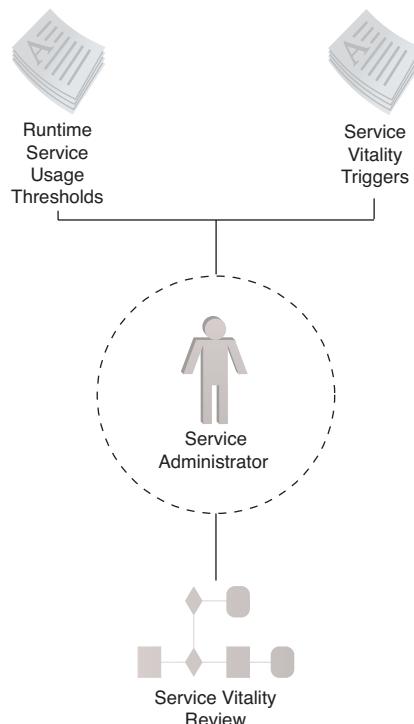


Figure 11.6

Service Usage and Monitoring governance precepts and processes associated with the Service Administrator role.

Related Precepts

- Runtime Service Usage Thresholds
- Service Vitality Triggers

Related Processes

- Service Vitality Review

Cloud Resource Administrator

For cloud-based services, the Cloud Resource Administrator will perform tasks similar to the Service Administrator, but, depending on the nature of the cloud environment, may be further required to take additional considerations into account.

For example, the following are common issues addressed by Cloud Resource Administrators acting on behalf of cloud consumer organizations that have deployed a service in a public cloud:

- runtime usage and performance fluctuations resulting from reliance on IT resources being shared by multiple cloud services from different cloud consumer organizations
- proprietary runtime service agents and monitoring tools provided by the cloud environment and perhaps not compatible with corresponding on-premise products
- automated or manual configuration of scaling characteristics of a service or of one or more of its underlying IT resources (in relation to scaling thresholds)
- billing limitations or options as they may apply, depending on the licensing or leasing model used by the cloud consumer organization

Furthermore, there may be unique security concerns that arise from making the service available via a cloud. These would be investigated by the Cloud Resource Administrator in collaboration with an SOA Security Specialist and/or a Cloud Security Specialist.

Related Precepts

- Runtime Service Usage Thresholds
- Service Vitality Triggers

Related Processes

- Service Vitality Review

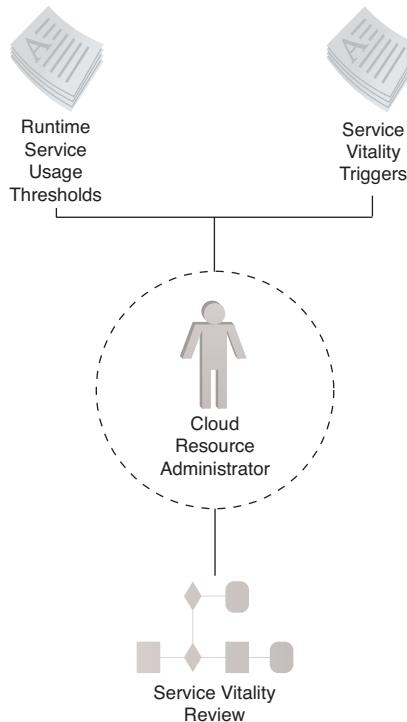


Figure 11.7

Service Usage and Monitoring governance precepts and processes associated with the Cloud Resource Administrator role.

Service Custodian

The Service Usage and Monitoring stage represents an on-going period of time during which the Service Custodian remains actively involved with any issues pertaining to the service's runtime behavior and performance, as well as its overall functional scope and integrity (in relation to how the purpose and functional context of the service was initially defined during the Service-Oriented Analysis stage). As with the Service Architect, this role takes part in all precepts and processes associated with this stage.

For cloud-based services, the Service Custodian, together with the Cloud Resource Administrator, will stay on top of any vitality-related issues that may arise (as a result of vitality triggers having executed). If significant changes are required to a service implementation, they will usually need to report the estimated impacts to the Cloud Service Owner for approval.

Related Precepts

- Runtime Service Usage Thresholds
- Service Vitality Triggers

Related Processes

- Service Vitality Review

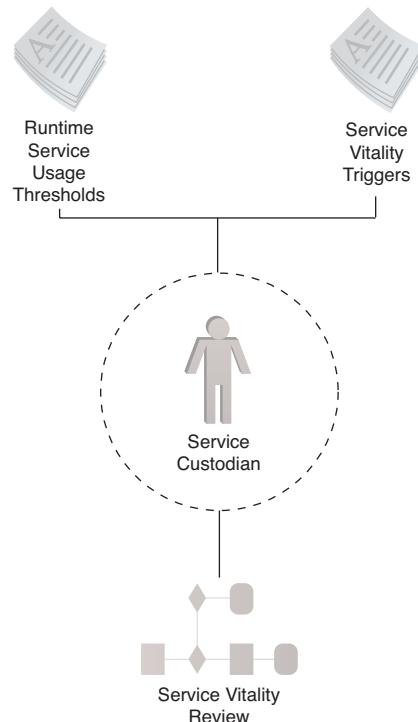


Figure 11.8

Service Usage and Monitoring governance precepts and processes associated with the Service Custodian role.

SOA Security Specialist

This role is pulled into precept definition and process reviews whenever security issues arise or when preventative measures need to be considered. For example, if a service's exception threshold is constantly exceeded, a vitality trigger may be executed resulting in a vitality review that could involve the SOA Security Specialist to help determine whether the increased number of recorded exceptions were the result of malicious service consumers carrying out periodic attacks on the service.

Related Precepts

- Runtime Service Usage Thresholds
- Service Vitality Triggers

Related Processes

- Service Vitality Review

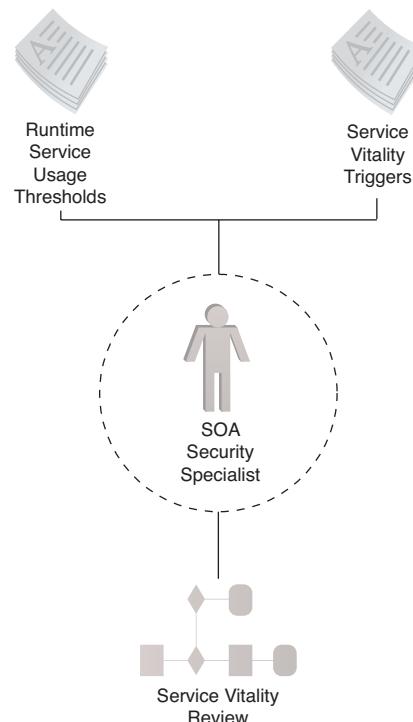


Figure 11.9

Service Usage and Monitoring governance precepts and processes associated with the SOA Security Specialist role.

SOA Governance Specialist

The SOA Governance Specialist will aid in defining and establishing the precepts associated with the Service Usage and Monitoring stage, and will further assist with carrying out vitality reviews. As explained in Chapter 13, SOA governance vitality can exist as a sub-framework to the overall SOA governance system. The definition and positioning of this framework (which encompasses vitality triggers, activities, and processes) are the responsibility of the SOA Governance Specialist.

Related Precepts

- Runtime Service Usage Thresholds
- Service Vitality Triggers

Related Processes

- Service Vitality Review

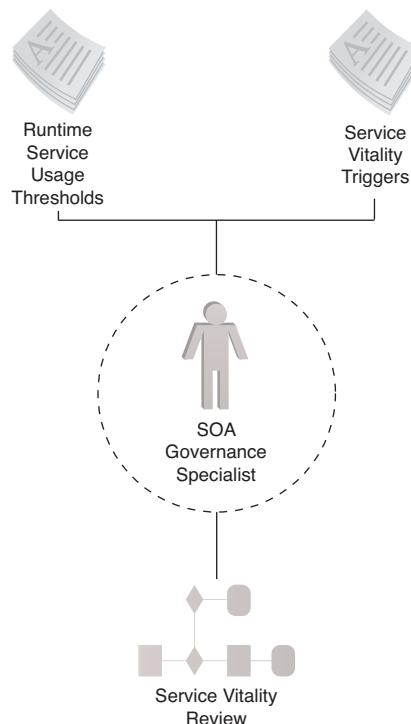


Figure 11.10

Service Usage and Monitoring governance precepts and processes associated with the SOA Governance Specialist role.

SUMMARY OF KEY POINTS

- Runtime thresholds can be set to help control and measure the usage of a service.
- Vitality triggers can be assigned to a given service in order to automate notification of the service status based on pre-defined criteria.
- Vitality reviews are carried out as a result of executed vitality triggers.

CASE STUDY EXAMPLE

The Raysmoore Product service has finally made its way into the Raysmoore production environment. As first explained in the *Case Study Example* section at the end of Chapter 8, this service was modeled to contain four variations of the Get service capability in order to accommodate business requirements specific to Raysmoore and Lovelt, while complying to the Service Normalization precept (Figure 11.11).

Figure 11.11

The Raysmoore Product service contract.



Specifically, the GetFull and GetRangeFull service capabilities allow for the retrieval of product inventory information for customers of Lovelt. The information provided includes current stock levels, which is something the corresponding Get and GetRange service capabilities do not provide, as per Raysmoore policy.

While deciding on threshold values for the application of the Runtime Service Usage Thresholds precept, the Service Architect, together with the SOA Governance Specialist, further examine these and related processing requirements of the four Get service capabilities. In addition to the general thresholds being applied to new services, they determine that the GetRangeFull service capability allows service consumers to issue queries that can return an unusually large amount of Product data (including historical and statistical data).

The Service Architect is concerned that some of the more complex queries could take a long time to execute and that the volume of returned data could put a strain on shared bandwidth, especially if the service is being accessed concurrently by multiple service consumers. As a result, they determine that the GetRangeFull service capability needs to be more strictly regulated than the Product service's other service capabilities, as follows:

- Whereas other Product service capabilities are assigned a Service Instance Threshold of 22, the GetRangeFull service capability is limited to spawning 10 service instances, when concurrently invoked.
- Similarly, the Service Data Throughput Threshold for the GetRangeFull service capability is set to half of what is allowed by the Product service's other service capabilities.

The Service Custodian makes note of these limitations in the Product service profile document and further warns that if usage demands exceed what the imposed thresholds allow, then a new version of this service capability may need to be developed, supporting a reduced query range in order to maintain higher thresholds.

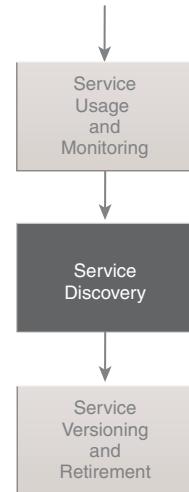
11.2 Governing Service Discovery

The dynamic of discovering reusable services for inclusion in new service-oriented solutions is fundamental to service-orientation and the realization of shared services, and perhaps one of the primary reasons that this stage has historically been most associated with SOA governance in general.

The following sections explore precepts and processes that help regulate the Service Discovery stage and, in particular, the use of the service registry.

NOTE

The service registry is an SOA governance technology explained in Chapter 14.



Precepts

Centralized Service Registry

IT enterprises can have multiple collections of services that exist as service inventories. For each well-defined collection, there should be a service registry, which is centralized so that it establishes itself as the sole source for official service discovery information.

When multiple domain service inventories exist, the following centralization rules usually apply:

- For every domain service inventory there should be only one central domain service registry.
- When a subset of services is shared across multiple domain service inventories, a central service registry can represent multiple service inventories or separate service registries can share a common set of data (most likely via replication).

SOA PRINCIPLES & PATTERNS

It is important to acknowledge that it is the application of the Service Discoverability (484) principle during the early design stages that helps make information published about a service both interpretable and discoverable. The Service Discovery stage then relies on these qualities to enable project teams to locate and understand upcoming and existing services when designing and assembling their new service-oriented solutions in compliance with governance regulations.

- When the shared subset of services corresponds cleanly to a service model (or some form of clearly distinguished service category), a separate service registry can be positioned as a central, cross-domain repository of service metadata specifically for that type of service.

This precept is applied to establish a centralized service registry and to mandate its use in support of the Service Discovery stage. Project teams delivering new services need to be required to have them recorded in the service registry in order to keep service registry data current and accurate.

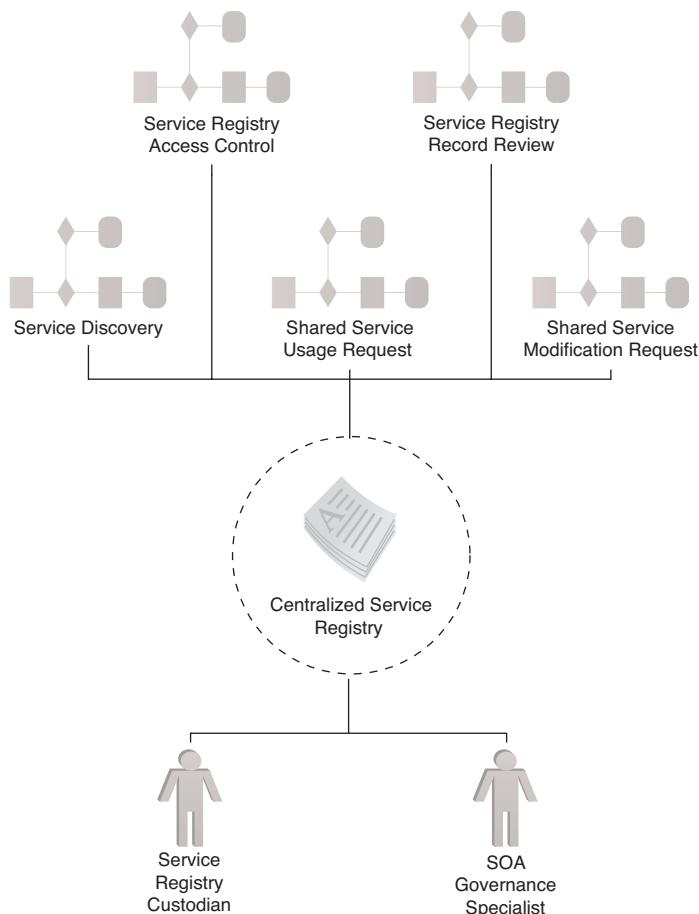


Figure 11.12

The Centralized Service Registry precept.

Related Processes

- Service Registry Access Control
- Service Registry Record Review
- Service Discovery
- Shared Service Usage Request
- Shared Service Modification Request

Related Roles

- Service Registry Custodian
- SOA Governance Specialist

SOA PRINCIPLES & PATTERNS

The Centralized Service Registry precept is based on the consistent application of the Metadata Centralization [536] pattern. The Cross-Domain Utility Layer [511] pattern defines an approach whereby reusable utility services can be shared across multiple domain service inventories.

Processes

Service Registry Access Control

A primary issue when governing Service Discovery is establishing the appropriate level of access to service registry records. An access control process enforcing specific rules and policies may need to be put in place to ensure that only authorized parties are able to locate and access some or all of the services or service capabilities within certain service registries.

SOA PRINCIPLES & PATTERNS

The Service Registry Access Control process relates to intentional limitations imposed by the application of the Service Abstraction (478) principle.

The following are sample access control considerations and rules:

- Some services can be selectively marked as “discoverable,” thereby making them undiscoverable to some users and groups. The need for this may be related to security requirements, but it can also be the result of service versioning in that a retired service may need to remain active to support existing service consumers (but its retired version should no longer be discoverable to new service consumers).
- If necessary, only a portion of service capabilities can be made discoverable or accessible. This may be required if a subset of service capabilities is intended only for certain types of service consumers.

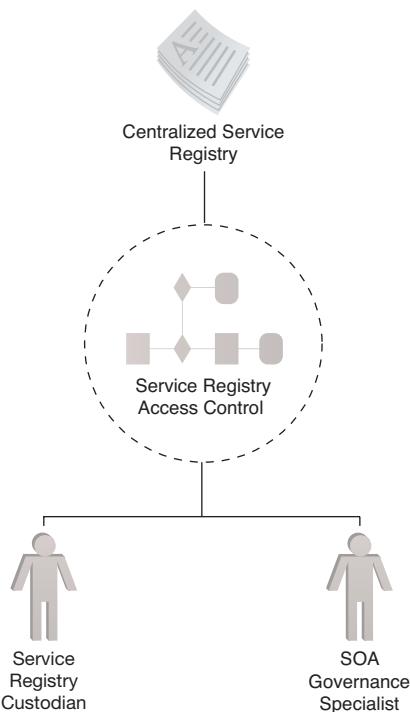


Figure 11.13
The Service Registry Access Control process.

Note that security controls used to limit access to a service's registry record are separate from any security mechanisms and requirements with which the service itself may have been designed. A service may be discoverable but still off-limits to those service consumers that do not possess the appropriate security credentials.

Related Precepts

- Centralized Service Registry

Related Roles

- Service Registry Custodian
- SOA Governance Specialist

NOTE

Although this precept raises security requirements, it does not directly relate to the involvement of the SOA Security Specialist role. The nature of the access control system that may need to be set up for a service registry is usually an internal administration matter and does not affect the security requirements of services or service-oriented solutions. However, if the service registry is made accessible outside of the organization boundary, then the SOA Security Specialist may need to participate with the application of this precept.

Service Registry Record Review

When adding to or updating a centralized service registry, a review process may be required to verify that new service registry records comply to all required authoring and technology standards (including access control rules). This review process is almost always carried out by the Service Registry Custodian together with the Technical Communications Specialist, but can also involve other roles, such as the SOA Governance Specialist and the Service Custodian.

Related Precepts

- Centralized Service Registry
- Service Metadata Standards (Chapter 12)

Related Roles

- Service Registry Custodian
- Service Custodian
- Technical Communications Specialist
- SOA Governance Specialist

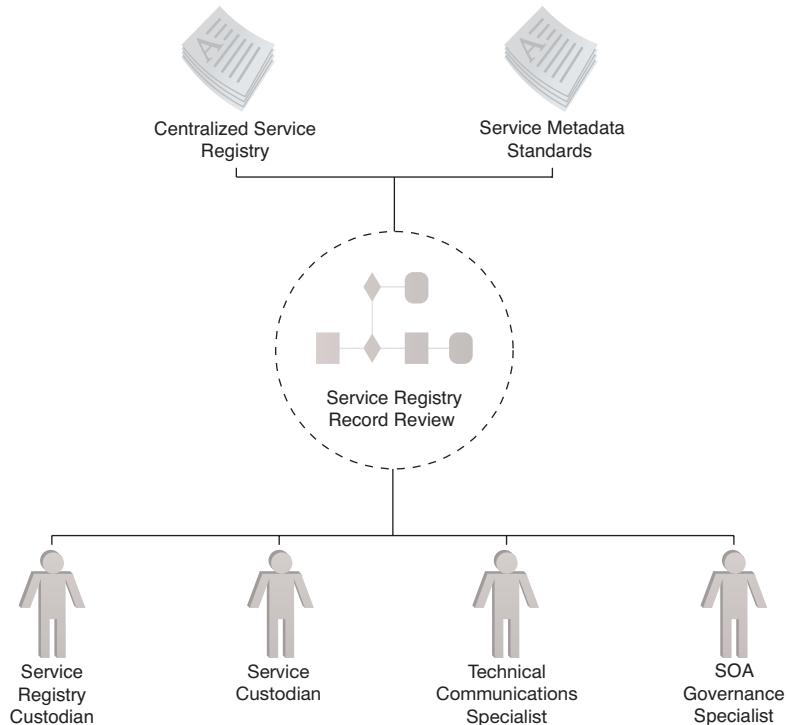


Figure 11.14

The Service Registry Record Review process.

Service Discovery

A formal discovery process needs to be in place, providing clear, step-by-step instructions for all project teams planning or actively delivering services and service-oriented solutions for a given service inventory.

The following is a sample discovery process that also includes steps pertaining to access control:

1. A project team member accesses a service registry and performs a search based on provided criteria.
2. Services marked discoverable for this user (or the group to which the user belongs) are searched and those with metadata matching the search criteria are returned.

3. The user chooses an agnostic service based on the provided service metadata and the request is routed to the corresponding Service Custodian.
4. The Service Custodian approves or rejects the request.
5. If access can be granted, a process to provision access to the actual service is invoked (and those involved with the service management are notified of the new service consumer).
6. If the request is rejected, the user is notified. Human contact with the Service Custodian may be required to address or clarify the reasons for the rejection.

Ensuring that service metadata within service registry records is authored consistently and in adherence to a common vocabulary and pre-defined standards is vital to the successful use of a service registry and the successful execution of the Service Discovery project stage.

The standards that govern service registry records are primarily concerned with guaranteeing that the metadata for a given service provides all of the keywords and other search criteria required for effective discovery queries and that, once discovered, the service metadata is easily understood and interpreted by those performing the searches.

Related Precepts

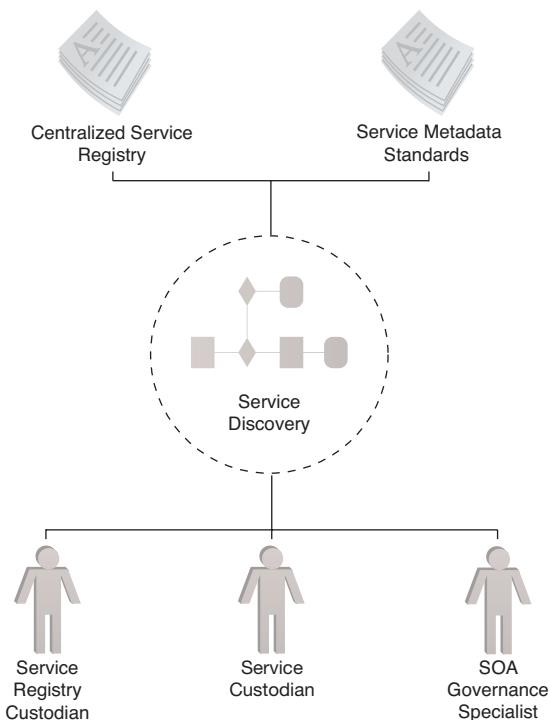
- Centralized Service Registry
- Service Metadata Standards (Chapter 12)

Related Roles

- Service Registry Custodian
- Service Custodian
- SOA Governance Specialist

SOA PRINCIPLES & PATTERNS

The Service Discovery process relates directly to the application of the Service Discoverability (484) principle, which encompasses metadata that is defined during the Service-Oriented Analysis and Service-Oriented Design stages, as well as any additional metadata authored for service registry records. In many ways, the success by which the Service Discovery stage is carried out by project teams can be traced back to the extent to which the Service Discoverability (484) principle was correctly applied during early analysis and design stages.

**Figure 11.15**

The Service Discovery process.

Shared Service Usage Request

When agnostic services with reusable logic are discovered by project teams wanting to share them within new service compositions, a formal process can be established to issue a request for reuse, along with information about how the service will be used within the new solution.

This type of process enables those overseeing the governance of a service to ensure that a given service implementation will not be stretched too thin or that a service will not be used in an inappropriate manner. Further, it allows for different service consumers to be prioritized based on the importance or urgency of each request.

For example, the request to reuse an agnostic service for a mission critical business automation requirement may receive a higher priority than other usage requests. In this case, a higher Service Instance Usage Threshold value may be assigned to the higher ranking service consumer.

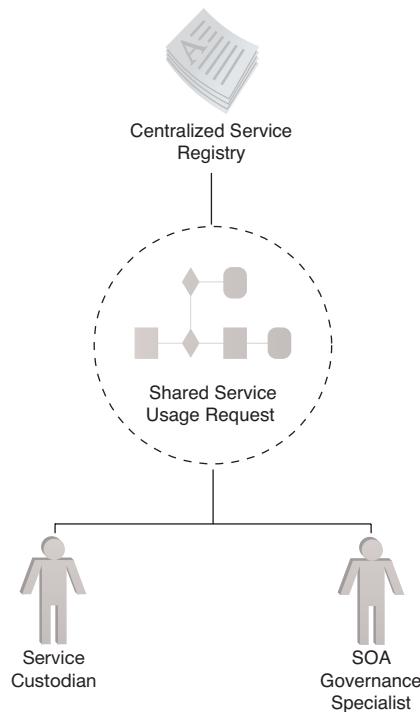


Figure 11.16

The Shared Service Usage Request process.

Related Precepts

- Service Registry

Related Roles

- Service Custodian
- SOA Governance Specialist

Shared Service Modification Request

There will commonly be situations when the project team requesting the use of a shared service will further request modifications to the service. This requires a separate process to address the following important considerations:

- The shared service contains reusable logic that will have typically been carefully designed based on a previously defined agnostic functional context. The integrity

of this context must be retained in order for the service to continue being an effective, shared enterprise resource.

- If a change or enhancement can be made to the service without compromising its agnostic functional context, then there will need to be an understanding of who will carry out this change and/or how the additional development effort will be funded.
- Functional changes to services (especially services already being actively shared) will almost always introduce a new major version. This will require that the modified service undergo the full delivery lifecycle plus be subject to existing version management precepts and processes.

When the custodian of a service (together with involvement from the SOA Governance Program Office) rejects a change request, it may alleviate the project team from having to use the discovered service. However, it may not provide them with the freedom to build the requested logic on their own.

Note that this type of request process may not necessarily be considered part of the Service Discovery stage. As it raises analysis and design concerns, it may be positioned as a pre-deployment process or as part of the Service Versioning and Retirement stage.

Related Precepts

- Centralized Service Registry

Related Roles

- Service Custodian
- SOA Governance Specialist

SOA PRINCIPLES & PATTERNS

Service Normalization [563] is a prime concern within any service inventory, and it requires that if the newly identified service logic is also deemed as reusable, that it either be placed in another shared service with the appropriate functional context or that it form the basis of a new agnostic functional context (requiring the delivery of a new reusable service).

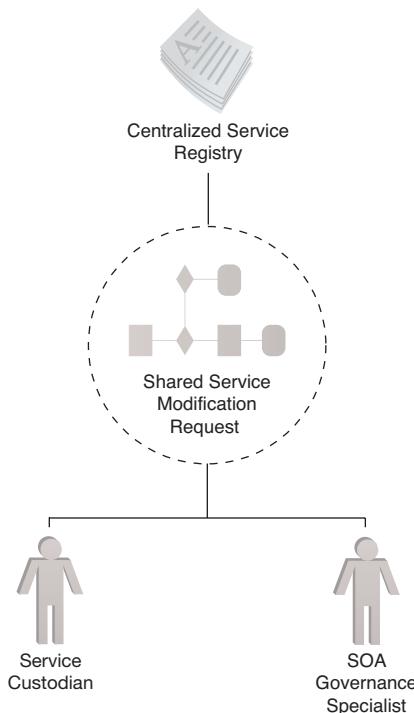


Figure 11.17
The Shared Service Modification Request process.

People (Roles)

Service Custodian

At the center of activity during the Service Discovery stage is the Service Custodian, who is a primary contact point for project teams attempting to locate, identify, and understand service metadata published for discoverability purposes. Further, the Service Custodian will often have the authority to determine whether new requests for different types of service usage are acceptable.

The Service Custodian is typically involved in all of the processes associated with this project stage, with the exception of the Service Registry Access Control process (unless access control rules specific to the custodian's service need to be defined).

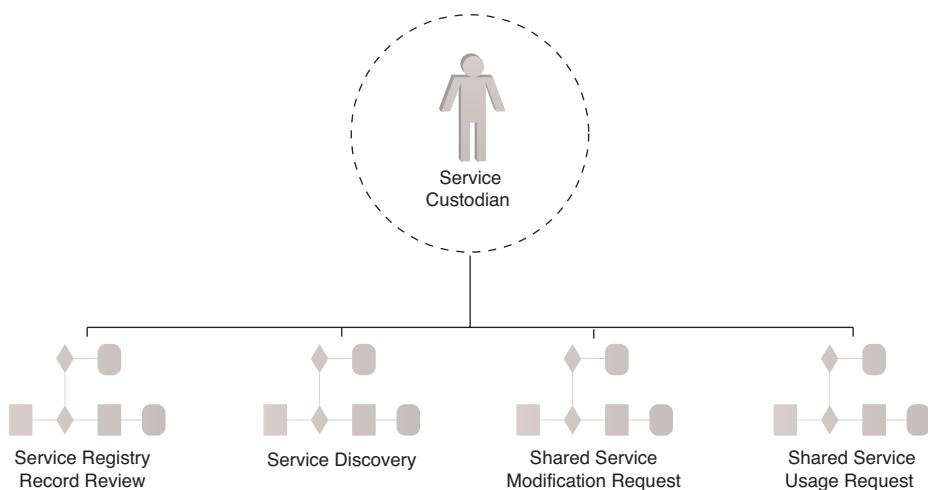


Figure 11.18

Service Discovery governance precepts and processes associated with the Service Custodian role.

Related Precepts

N/A

Related Processes

- Service Registry Record Review
- Service Discovery
- Shared Service Modification Request
- Shared Service Usage Request

Service Registry Custodian

All governance tasks that involve the service registry during the Service Discovery project stage will also involve the registry's custodian. This role will lead the effort to establish centralized service registries and further acts as a principal participant in the Service Registry Access Control and Service Registry Record Review processes. In support of the Service Discovery process, the Service Registry Custodian assumes more of an advisory role.

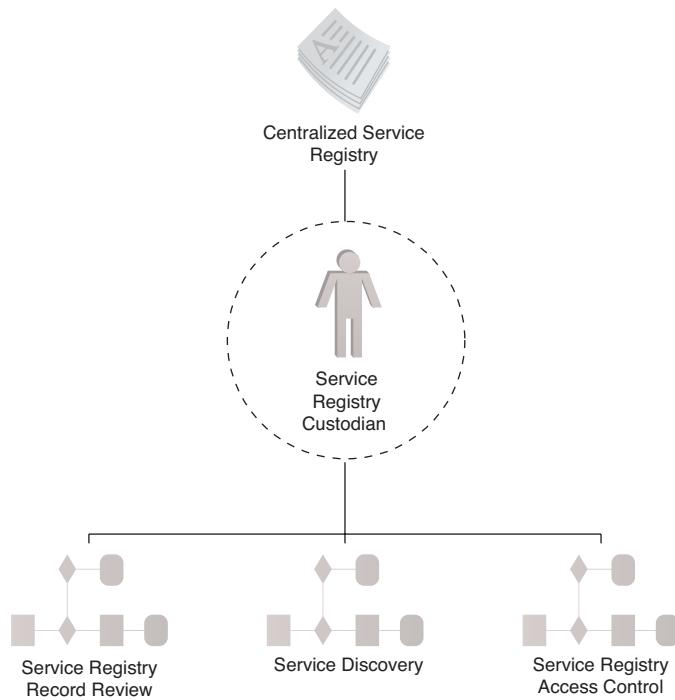


Figure 11.19

Service Discovery governance precepts and processes associated with the Service Registry Custodian role.

Related Precepts

- Centralized Service Registry

Related Processes

- Service Registry Access Control
- Service Registry Record Review
- Service Discovery

Technical Communications Specialist

With the responsibility of ensuring the communications quality of service metadata authored and published in support of the Service Discovery stage and the Service Discovery process, the Technical Communications Specialist is an expected participant in the Service Registry Record Review process.

Related Precepts

N/A

Related Processes

- Service Registry Record Review

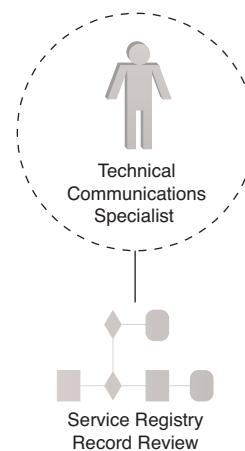


Figure 11.20

Service Discovery governance precepts and processes associated with the Technical Communications Specialist role.

SOA Governance Specialist

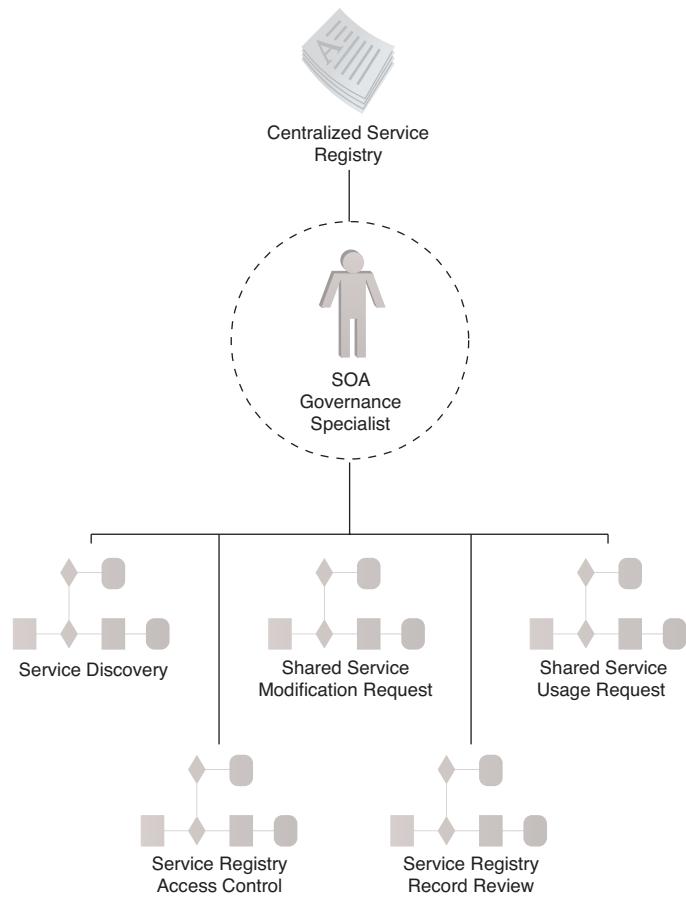
The SOA Governance Specialist works closely with the Service Registry Custodian to help establish the Centralized Service Registry precept and related processes, and will further aid the Service Custodian (and other roles) with governance processes and activities pertaining to the discovery of agnostic services.

Related Precepts

- Centralized Service Registry

Related Processes

- Service Registry Access Control
- Service Registry Record Review
- Service Discovery
- Shared Service Modification Request
- Shared Service Usage Request

**Figure 11.21**

Service Discovery governance precepts and processes associated with the SOA Governance Specialist role.

SUMMARY OF KEY POINTS

- The success of the Service Discovery stage is primarily dependent on the quality of the metadata recorded in the service registry. This is why there are several governance precepts and processes concerned with service metadata definition.
- The importance of service metadata pertains both to the discoverability of the service during the Service Discovery stage, as well as the interpretability of the discovered metadata by a range of project team members. These characteristics are instilled within service contracts and service registry data via the application of the Service Discoverability (484) principle during the Service-Oriented Design stage.

CASE STUDY EXAMPLE

During the original definition and implementation of the Productions and Operations service inventory architecture and supporting infrastructure, the Raysmoore project team applies the Centralized Service Registry precept (Table 11.1) previously established by the SOA Governance Program Office.

In support of the Centralized Service Registry precept, the SOA Governance Program Office introduces a custom variation of the Service Registry Access Control process that limits access to service metadata based on pre-defined rules. These access control rules require that the service registry be split into three principal sections:

- the *public* section that is open to external business partners
- the *internal* section that is restricted to Raysmoore and its subsidiaries
- the *restricted* section that is limited to individuals on project teams building service consumer programs authorized for a given service

The SOA Governance Program Office also helps create a formal certification process for “on-boarding” new consumers of each service (which leads to an additional precept and process).

Centralized Service Registry Precept	
<p><i>Objective:</i> Each service inventory must have one central service registry used to record metadata for all services within the service inventory.</p>	
<p><i>Policy:</i> Ensure that a common metadata vocabulary is used.</p>	<p><i>Policy:</i> Ensure that standardized access control rules are used.</p>
<p><i>Standard:</i> The metadata vocabulary must incorporate and be in alignment with existing service candidate and service naming conventions. Therefore, this precept must be applied together with the Service Metadata Standards precept (described separately in Chapter 12).</p>	<p><i>Standard:</i> Pre-defined security groups or categories must be used for all service registry records. Custom variations of pre-defined access control rules must be subject to a review and application for a waiver.</p>
	<p><i>Guideline:</i> When carrying out Service Discovery, this precept can be enforced by a manual or automated Service Registry Access Control process, depending on the preference of the project team members requesting access to protected service registry records. A manual process may be required when project teams need to request information about registered services that are hidden from discovery searches. In this case, the Service Registry Custodian will need to manually administer the request and, if approved, manually provide the requested service metadata.</p>
<p><i>Standard:</i> Require that the service registry is configured to limit access control to pre-defined groups and to further establish parameters that support the standardization of metadata, as per the conventions defined in the Service Metadata Standards precept.</p>	

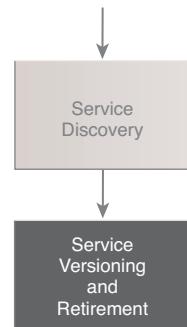
Table 11.1

The Centralized Service Registry precept.

11.3 Governing Service Versioning and Retirement

Service versioning represents the most evolutionary stage in a service's lifecycle. When the need for a new version of a service emerges, clear rules and regulations are required to ensure that whatever changes are introduced by the new service version do not negatively impact or disrupt other services and service consumers within the service inventory—especially those that already have dependencies on the updated service.

Similarly, the actual termination or deactivation of a service or service version is another natural part of a service's overall lifecycle, and an occurrence for which governance attention needs to be planned to further ensure no inadvertent disruption. The precepts in this section help establish fundamental regulatory controls to establish a structured system for service versioning and retirement in support of an IT enterprise's configuration management preferences.



Precepts

Service Versioning Strategy

The most important aspect of performing service versioning is that a set of rules is in place to ensure that each service within a given service inventory is versioned consistently. Versioning rules are primarily concerned with how services can be versioned in response to compatible or incompatible changes, as they relate to backwards and forwards compatibility.

These rules form the basis of a versioning strategy, of which three common types exist:

- *Strict* – Any compatible or incompatible changes result in a new version of the service contract. This approach does not support backwards or forwards compatibility.
- *Flexible* – Any incompatible change results in a new version of the service contract and the contract is designed to support backwards compatibility but not forwards compatibility.
- *Loose* – Any incompatible change results in a new version of the service contract and the contract is designed to support backwards compatibility and forwards compatibility.

NOTE

Appendix F explores each of these strategies in more detail, and further explains the differences between compatible and incompatible changes, as well as backwards and forwards compatibility.

Each service versioning strategy approaches the governance of the Service Versioning stage in a distinct manner. Therefore, each service versioning strategy will tend to further establish a set of distinct precepts that address more granular service versioning issues, such as:

- service version identification and labeling within service contracts and service registry records
- moderating the number of service versions that are allowed to exist concurrently in the production environment
- establishing when service consumers are required to switch to a new service version, even when this change impacts the service consumer design

Further precepts pertaining to the versioning of service logic changes and changes to the underlying service implementation and resources can also be defined as part of the overall versioning strategy.

Related Processes

- Service Versioning

Related Roles

- Enterprise Design Standards Custodian
- Schema Custodian
- Policy Custodian
- SOA Governance Specialist

SOA PRINCIPLES & PATTERNS

Depending on the nature of the precepts, various patterns can be involved, including:

- Compatible Change [505]
- Version Identification [575]
- Service Refactoring [565]
- Service Decomposition [556]
- Proxy Capability [547]

Note that some of these patterns relate to service contract versioning, while others are focused on versioning requirements resulting from service logic changes.

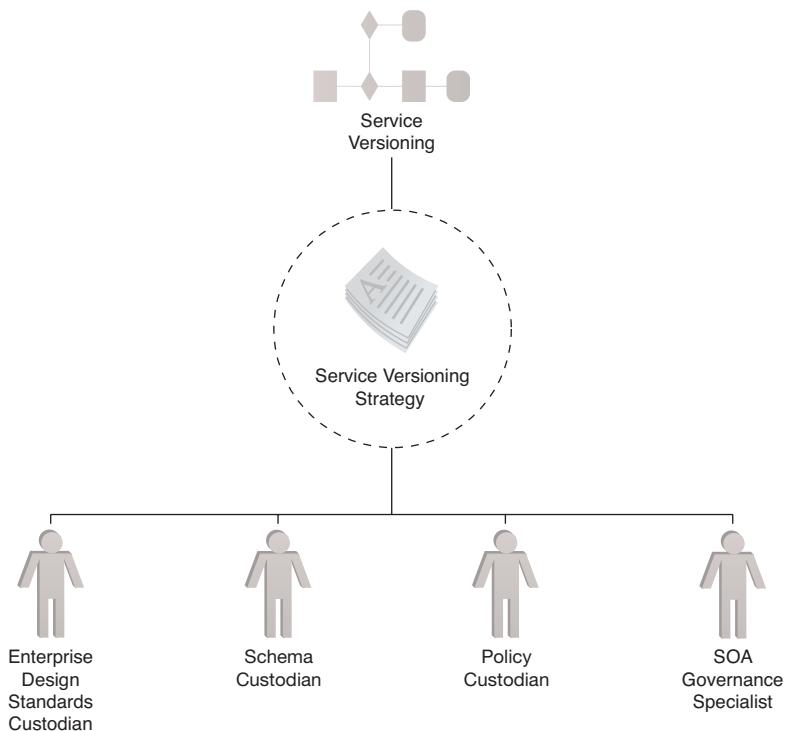


Figure 11.22

The Service Versioning Strategy precept.

SLA Versioning Rules

In addition to the rules defined in the chosen service versioning strategy, additional versioning rules and preferences can be added for individual services. Most commonly, these rules are expressed in human-readable language as part of a service's SLA.

Examples of such rules include:

- The maximum number of versions of a service that can be supported at any one time.
- How the consumers of a service are notified when the service version is changed and/or about to become unsupported.
- The predefined period of time that service consumers of a retiring service version have to migrate to the new version.

- When and how access to unsupported or retired service versions will be revoked.
- How to access a test environment, together with realistic simulated data, that will be made available to allow service consumers to test new service versions before they migrate their production systems to use them. This applies even when the new versions are expected to be backwards compatible.

These and other types of rules can also be part of the overarching service versioning strategy.

NOTE

Various forms of individual service versioning rules can be expressed in technical policy definitions instead of SLAs. One such example is the use of ignorable or optional WS-Policy assertions. Policy-related governance topics are covered in Chapter 12. Policy assertions are explained in the book *Web Service Contract Design & Versioning for SOA*.

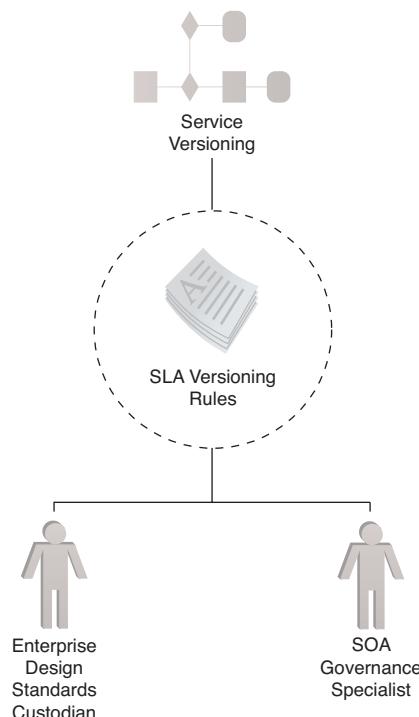


Figure 11.23
The SLA Versioning Rules precept.

Related Processes

- Service Versioning

Related Roles

- Enterprise Design Standards Custodian
- SOA Governance Specialist

Service Retirement Notification

A standardized mechanism or system needs to be in place in order to communicate the pending and completed retirement of a service. This mechanism must issue notifications to raise awareness of the service termination, beyond the mere change of the service status within the central service registry.

These notification requirements, along with the mechanisms used to carry out the notification, are standardized by this precept so that communication of pending and past service terminations remains consistent throughout a service inventory.

SOA PRINCIPLES & PATTERNS

An SOA design pattern developed specifically in support of the Service Retirement Notification precept is Termination Notification [569], which can be applied by adding an ignorable WS-Policy assertion in a service contract to indicate the scheduled retirement date of the service.

Related Processes

- Service Retirement

Related Roles

- Service Administrator
- Cloud Resource Administrator
- Enterprise Design Standards Custodian
- SOA Governance Specialist

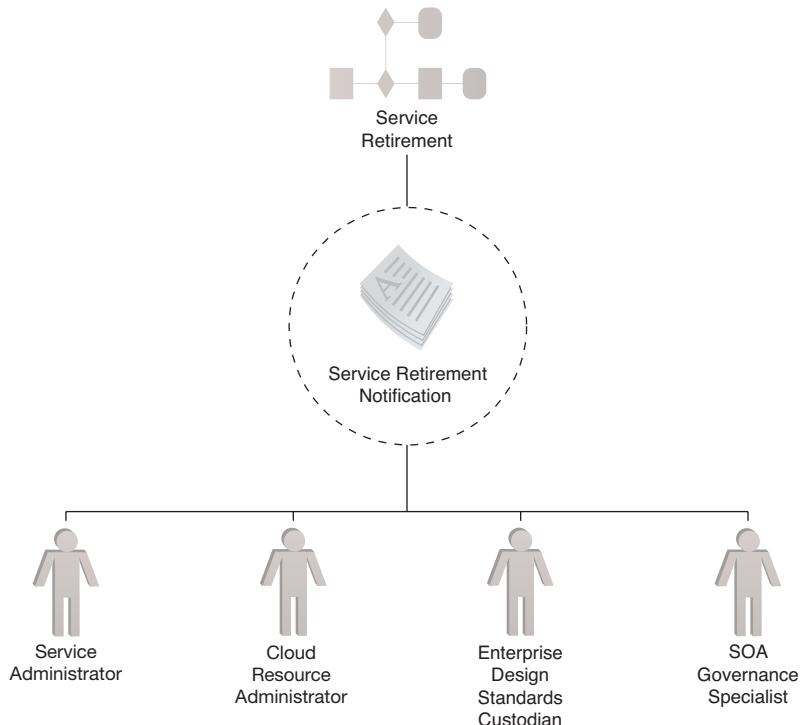


Figure 11.24

The Service Retirement Notification precept.

Processes

Service Versioning

With the chosen service versioning strategy comes one or more formal service versioning processes that establish step-by-step procedures for phasing new versions of services into a service inventory. These processes may trigger other project stages (or parts of project stages), such as testing and deployment. Generally, these stages are further customized to accommodate various factors.

For example, the new service version may need to:

- be deployed alongside one or more older service versions
- immediately support a number of service consumers that have so far been using the older version of the service
- introduce new logic and capabilities that require additional testing effort as well as new forms of tests

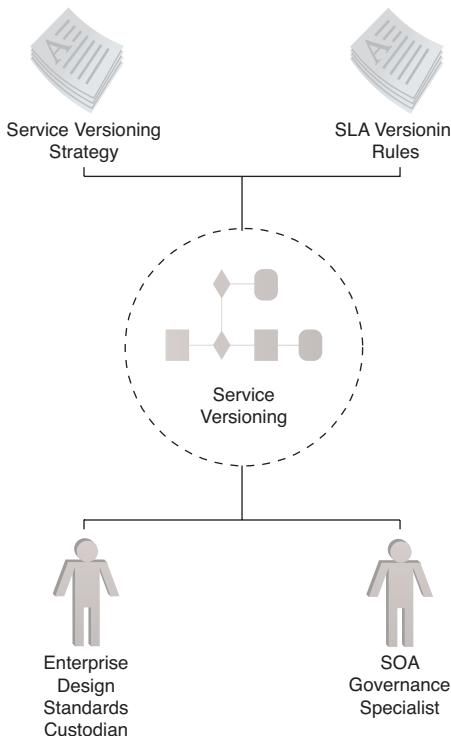


Figure 11.25
The Service Versioning process.

Service versioning processes will further include review steps to ensure that the service has complied with all requirements prior to being deployed into the production environment.

Related Precepts

- Service Versioning Strategy
- SLA Versioning Rules

Related Roles

- Enterprise Design Standards Custodian
- SOA Governance Specialist

Service Retirement

The retirement of a service that has been actively in use for an extended period of time is an important and formal process that needs to be defined and administered to ensure that the planned termination occurs without disruption. Often this is possible with proper advance planning because the actual termination date of the service can usually be pre-determined.

The Service Retirement process includes steps that carry out the Service Retirement Notification precept, but the primary purpose of this process is to investigate all possible (past and current) dependencies upon the to-be-terminated service.

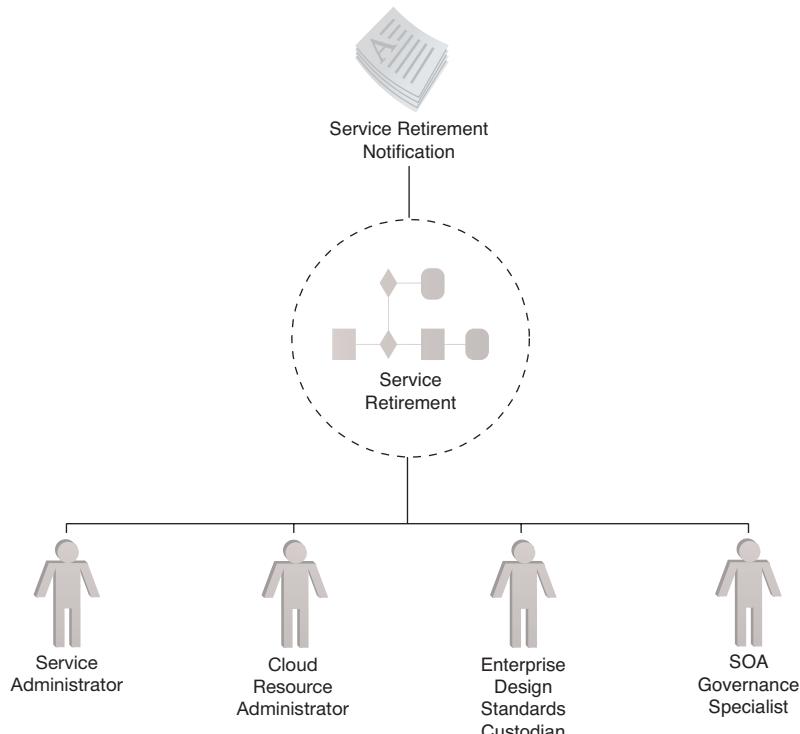


Figure 11.26

The Service Retirement process.

For example:

- service consumers that currently or periodically access and use the service
- service composition architectures that encompass the service architecture
- shared resources and legacy systems that are being accessed by the service (and may therefore be behaviorally impacted by the service's absence)

Furthermore, project teams that may be planning or actively designing new consumer programs for the service may need to be located and notified.

Related Precepts

- Service Retirement Notification

Related Roles

- Service Administrator
- Cloud Resource Administrator
- Enterprise Design Standards Custodian
- SOA Governance Specialist

People (Roles)

Enterprise Design Standards Custodian

Process and precepts that pertain to the Service Versioning and Retirement stage are primarily based on standards and conventions that need to be established to ensure consistency in how the logic and contracts of services within a given service inventory are evolved and then terminated. This requires the involvement of the Enterprise Design Standards Custodian to help define these standards and strategies and to further ensure they are created in alignment with and in support of other design standards.

Related Precepts

- Service Versioning Strategy
- SLA Versioning Rules
- Service Retirement Notification

Related Processes

- Service Versioning
- Service Retirement

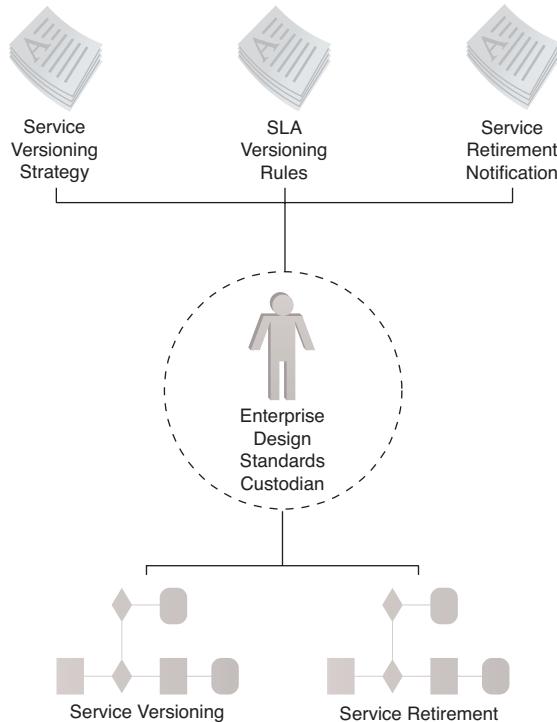


Figure 11.27

Service Versioning and Retirement governance precepts and processes associated with the Enterprise Design Standards Custodian role.

Service Administrator

The Service Retirement Notification precept and the Service Retirement process collectively regulate the termination of an active service. The Service Administrator can be a primary contributor to establishing these governance controls to ensure that they are defined in compliance with production runtime environments and affected IT resources.

Related Precepts

- Service Retirement Notification

Related Processes

- Service Retirement



Figure 11.28

Service Versioning and Retirement governance precepts and processes associated with the Service Administrator role.

Cloud Resource Administrator

There will likely be special considerations that need to be addressed by the Service Retirement Notification precept and the Service Retirement process for cloud-based services. For example, there may be licensing and billing arrangements that require termination or deferral to different cloud services. The cloud provider organization may require advance notice of a pending cloud service retirement and if the cloud service has been accessible to multiple external cloud consumer organizations (as it would be as part of an SaaS offering), a more elaborate notification system may be required. These and other issues are the responsibility of the Cloud Resource Administrator.

Related Precepts

- Service Retirement Notification

Related Processes

- Service Retirement



Figure 11.29

Service Versioning and Retirement governance precepts and processes associated with the Cloud Resource Administrator role.

Schema Custodian

Part of the Service Versioning Strategy precept includes strategies and standards that govern the versioning of XML schemas and data models. For this subset of the precept, the Schema Custodian can provide input and recommendations.

Related Precepts

- Service Versioning Strategy

Related Processes

N/A

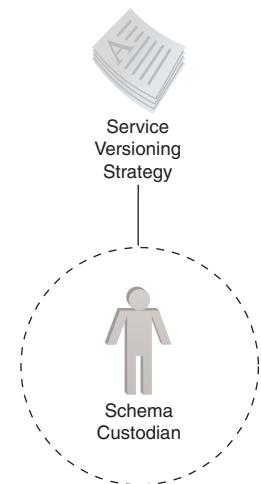


Figure 11.30

Service Versioning and Retirement governance precepts and processes associated with the Schema Custodian role.

Policy Custodian

As with the involvement of the Schema Custodian, the Policy Custodian can provide guidance for the versioning of technical business and operational policies, especially in relation to considerations raised by the Policy Centralization precept (as described in Chapter 12).

Related Precepts

- Service Versioning Strategy

Related Processes

N/A

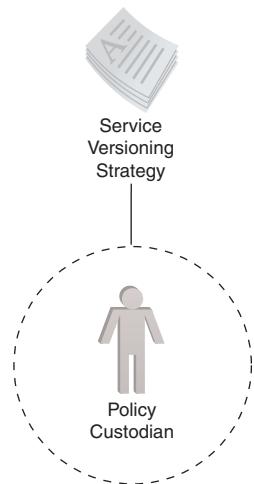


Figure 11.31

Service Versioning and Retirement governance precepts and processes associated with the Policy Custodian role.

SOA Governance Specialist

The effort required to establish the precepts and processes that govern the versioning and termination of services may fall squarely on the shoulders of the SOA Governance Specialist. Assistance from the Enterprise Design Standards Custodian will be required, but this role may also involve other specialists as needed.

Related Precepts

- Service Versioning Strategy
- SLA Versioning Rules
- Service Retirement Notification

Related Processes

- Service Versioning
- Service Retirement

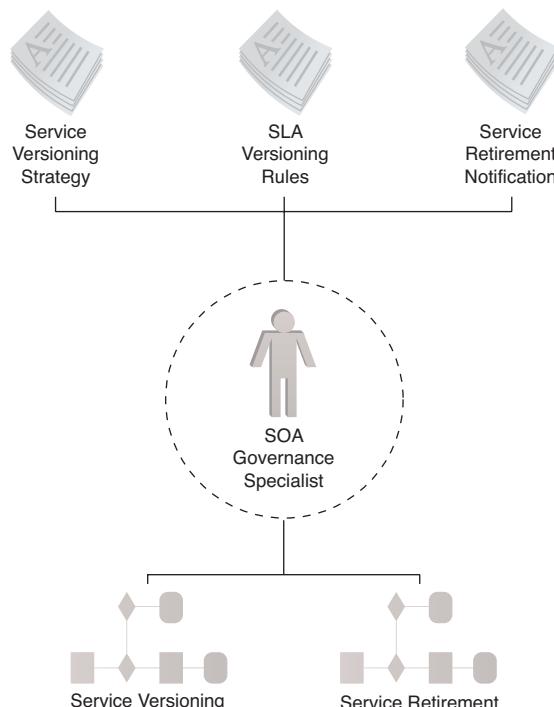


Figure 11.32

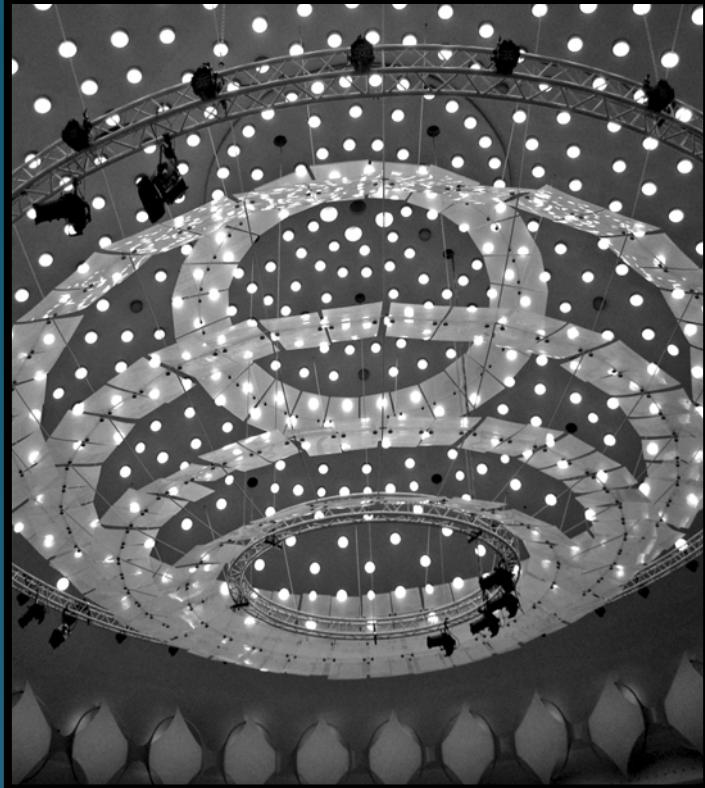
Service Versioning and Retirement governance precepts and processes associated with the SOA Governance Specialist role.

SUMMARY OF KEY POINTS

- The Service Versioning Strategy precept requires that a service inventory-wide service versioning system be established. Additional, more detailed precepts are then defined, specific to the overarching strategy.
 - In addition to a Service Versioning process, there is the need for a Service Retirement process that is further supported by a formal Service Retirement Notification precept.
-

NOTE

Examples pertaining to the Service Versioning Strategy precept are provided in Appendix F. Note that these are technical examples that display code for Web services and REST services.



Part III

Strategic Governance

Chapter 12: Service Information and Service Policy Governance

Chapter 13: SOA Governance Vitality

Chapter 14: SOA Governance Technology

This page intentionally left blank



Chapter 12

Service Information and Service Policy Governance

12.1 Overview

12.2 Governance Controls

12.3 Guidelines for Establishing Enterprise Business Models

SOA PRINCIPLES & PATTERNS REFERENCED IN THIS CHAPTER

- Messaging Metadata [535]
- Policy Centralization [543]
- Service Abstraction (478)
- Validation Abstraction [574]

The adoption of service-orientation breaks down silos within the organization by establishing a layer of normalized functional boundaries, each represented by a separate service. So far in this book we've been focusing a great deal on the governance issues that pertain to this service layer as they relate to individual stages of the SOA project lifecycle.

In this chapter we focus on the governance requirements of data exchanged by services, its origin, its definition, and its usage to further add definition to the data and its relationships with services.

NOTE

This chapter is primarily focused on business data related to business-centric services, such as entity and task services. For a description of how business services differ from non-business services (such as utility services), see the *Service Models* section in Chapter 3.

Also, unlike Chapters 7 to 11 where precepts, processes, and roles were organized according to project stages, in this chapter they are grouped into dedicated sections, as each may apply to one or more stages.

12.1 Overview

This chapter is about the governance of service information and service policies. Service information governance is the practice of identifying and evolving governance controls that ensure the provision of timely and accurate information. Service policy governance is focused on the regulation of operational and business policies that can be machine- or human-readable.

Let's begin by clarifying the key terminology used in the upcoming sections, namely the terms *data*, *information*, and *policy*.

Service Data vs. Service Information

The first step to understanding the scope and purpose of service information governance is to make a clear distinction between data and information. In the science of knowledge management, there is an established concept referred to as the *knowledge hierarchy* and also known as the *DIKW hierarchy*.

The acronym DIKW represents the following:

- *Data* – raw facts and symbols without context and meaning
- *Information* – data given meaning by placing it within a context
- *Knowledge* – information that is understood, making it useful and actionable
- *Wisdom* – experience applied to knowledge, providing the ability to make appropriate decisions

How these parts of the hierarchy relate to each other is illustrated in Figure 12.1.

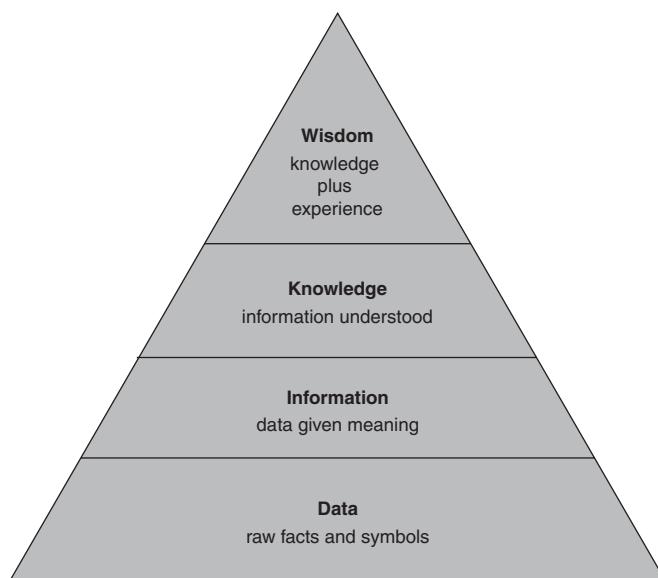


Figure 12.1

The DIKW pyramid shows how raw facts and figures (data) are the foundation for ultimately understanding and being able to (wisely) act upon information.

Within this chapter only, we will use the terminology established by the DIKW hierarchy to make a distinction between *data* (raw facts and symbols) and *information* (data given meaning). When defining and using business-centric services (as opposed to utility services), we are more concerned with information rather than just data. This is because each business service's functional boundary is derived from a source of business intelligence (such as a business entity or a business process) that gives the data meaning (hence our use of the term “information”).

Business services provide service consumers information in the form of messages associated with business intelligence. This is how business services place data in context, giving it meaning. Because of this ontological commitment made by business services, an SOA governance system must address the governance of both the data and its meaning. This is because SOA governance precepts can help ensure that services provide this information accurately and that service consumers can trust this information without requiring knowledge of its origins.

Policies 101

Just about any organization works with policies that provide parameters and guidance for how tasks and responsibilities need to be carried out. Some of these requirements are mandated by external organizations (such as government agencies or industry standards bodies), while others may be imposed by internal organizational entities (such as accounting, human resources, or legal departments).

In general, policies are stated as real-world requirements or rules. For example:

- An order cannot be shipped until payment has been received.
- An employee cannot work more than 40 hours per week without approval.
- Each aircraft engine needs to be inspected after every 50 hours of flying.

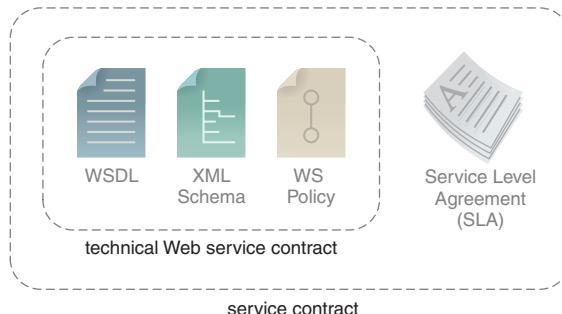
Policies have traditionally been published as human-readable (or natural language) documents. Such policies therefore rely on manual processes, carried out by humans, to ensure enforcement. Failing to enforce policies can lead to serious consequences, with potential legal, financial, and operational ramifications. Because manual enforcement by humans can be prone to human error and inconsistency, it can introduce risk.

As a result, organizations have sought ways to automate policy enforcement via the use of machine-readable (or technical) policies. Technologies, such as WS-Policy (Figure 12.2) and policy management products (explained in Chapter 14) have emerged to provide various features for technical policy definition and runtime enforcement.

Although the substance and content quality of individual policies can vary, technical policies tend to be more focused on processing data and human-readable policies are generally authored with more information (data with meaning). Depending on the nature of the policy content and its purpose and intended consumers, different types of policies can be authored. The precepts described in this chapter focus on business and operational policies, each of which can exist in human-readable or machine-readable format.

Figure 12.2

A WS-Policy definition as part of a Web service contract.



Regardless of their type, technical policies can be associated with services as an extension to the service contract or as part of the underlying service logic. Using a technical language (such as WS-Policy), machine-readable policies can express service contract level constraints and qualities that relate to and express service behavior and requirements.

Although it is possible and sometimes required, it is less common to embed policy processing logic as part of the service logic. Embedded policy logic can be difficult to maintain, especially when one policy applies to multiple services. Furthermore, having technical policies defined as part of the service contract makes the policy requirements clear in advance to service consumer designers.

SOA PRINCIPLES & PATTERNS

An SOA design pattern that advocates placing less validation logic in the service contract is Validation Abstraction [574]. This pattern is applied primarily in support of the Service Abstraction (478) principle that aims to minimize what is published about a service in order to foster positive coupling between services and service consumers. The usage of the Validation Abstraction [574] pattern may lead to justification for locating some forms of policy processing within the service logic.

NOTE

The policies discussed in this chapter are focused exclusively on service contract-related policy definitions. However, the associated technologies used to create policy definitions are beyond the scope of this book.

Chapters 10, 16, and 17 in the book *Web Service Contract Design & Versioning for SOA* are dedicated to fundamental and advanced topics pertaining to the use of WS-Policy related syntax and technologies.

12.2 Governance Controls

A core objective of service-oriented computing is to achieve a state of intrinsic interoperability among software programs delivered as services. A baseline requirement for doing so is ensuring that the data exchanged is understood by services and their consumers. To help foster an understanding of the meaning of service data, governance precepts are required to position and regulate service information and policy content.

Precepts

Enterprise Business Dictionary/Domain Business Dictionary

When integrating disparate and silo-based applications, the incompatibility of data meaning has historically caused problems. For example, when different systems colloquially refer to a “claim” without further qualification, we can end up with different meanings being associated to this term. One system may deal with property insurance claims, while the other processes employee health insurance claims. These types of issues have historically been resolved with traditional integration architectures, whereby point-to-point integration channels were created between each system. Each channel contained custom programming comprised of data mapping and transformation logic that was executed at runtime, each time data exchange needed to occur.

As previously explained, one of the primary goals of service-orientation is to avoid having to resort to integration architectures. However, suppose, within a services environment, two disparate service consumers become aware of and decide to use a Claim Submission service. With no additional information about the service (or its information) available, neither service consumer may have a reason to believe that “claim” means anything other than what it understands it to be. This precept aims to avoid this scenario by requiring that a business dictionary be established.

There are two variations of this precept:

- Enterprise Business Dictionary
- Domain Business Dictionary

The difference is a matter of scope. An Enterprise Business Dictionary applies to the IT enterprise as a whole. It mandates the creation of a central dictionary to be used by all services within all service inventories. The Domain Business Dictionary precept requires that business dictionaries focus on specific subsets of the IT enterprise. The most common application of this precept is to establish a business dictionary with a scope equal to a particular domain service inventory.

The reason separate precepts exist is because they can be used together. For example, an Enterprise Business Dictionary may be developed to regulate a common set of business terms (such as those that perhaps represent common business documents), but it may not include definitions for all possible business terms. Additional Domain Business Dictionaries can be authored to standardize domain-specific terminology, even if it results in some terms having different definitions.

Either way, each business dictionary provides an official and centralized resource for the definition of information. Each term within the dictionary is given an individual meaning, and qualifiers (such as “medical” claim) are used to give terms more specific (yet still individual) meanings.

Although the creation of a business dictionary (or the updating of an existing business dictionary) is a common initial step during SOA adoption projects, its usage is by no means limited to service information or to the SOA adoption scope.

NOTE

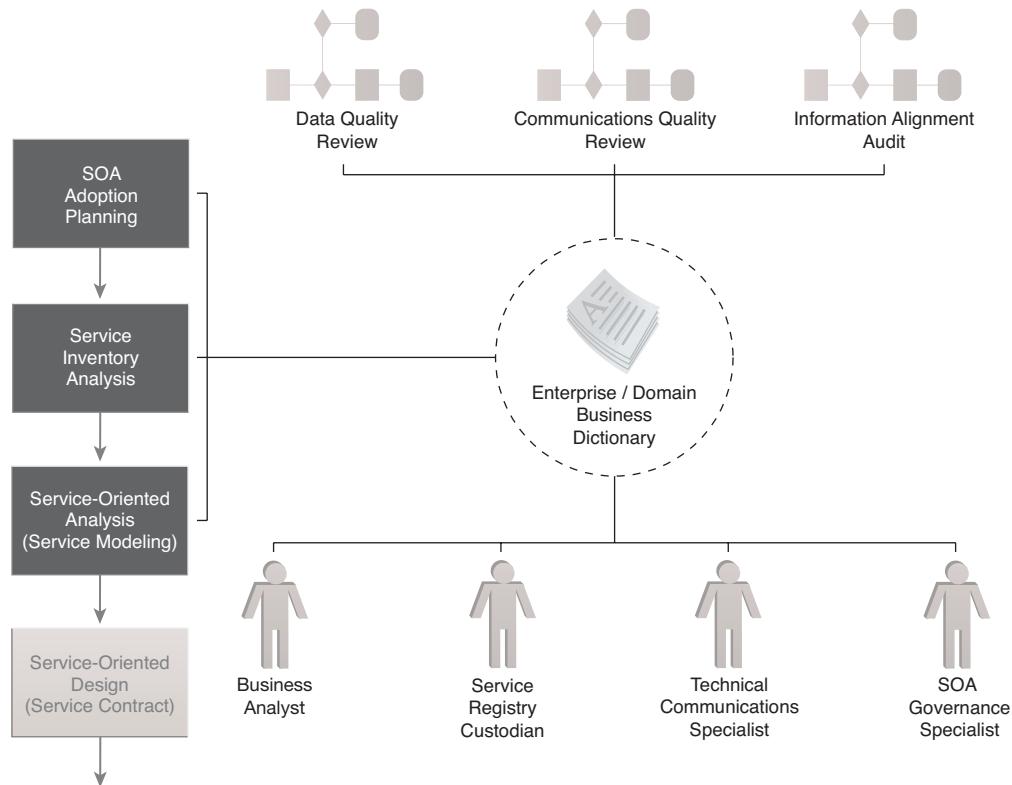
A business dictionary is different from a data dictionary. Whereas the latter is focused on the definition of data residing within database tables, the former aims to establish a “single source of truth” for *information* so as to foster a common business understanding.

Related Processes

- Data Quality Review
- Communications Quality Review
- Information Alignment Audit

Related Roles

- Business Analyst
- Service Registry Custodian
- Technical Communications Specialist
- SOA Governance Specialist

**Figure 12.3**

This precept can be applied as early as the SOA Adoption Planning stage, but also throughout the Service Inventory Analysis and Service-Oriented Analysis stages.

Service Metadata Standards

Metadata is typically defined as “data about data.” But given the terminology we established earlier in this chapter, it can more accurately be defined as “information about data.” Essentially, metadata is information about data of importance to an organization. Those responsible for designing and administering databases have long catalogued technical metadata that describes databases, tables, columns, and other aspects of physical data management systems. However, functional metadata—descriptions of business meaning and relevancy of data—is less commonly captured and documented.

Metadata, as it pertains to services and information exchanged by services, is very much focused on establishing functional meaning and context.

Established forms of service metadata include:

- *Technology Information* – Metadata that describes the technical implementation of the underlying service logic.
- *Functional Information* – Metadata that describes what the service is capable of.
- *Programmatic Logic Information* – Metadata that describes how the service carries out its capabilities.
- *Quality of Service Information* – Metadata that describes service behavior, limitations, and interaction requirements.

Depending on the implementation medium used to build services, service metadata standards can be encompassed within the Service Contract Design Standards precept (explained in the *Governing Service-Oriented Design* section of Chapter 9). However, if the scope of an organization's overall information governance initiative is beyond that of any one SOA adoption project, metadata standards can exist independently and then referenced by Service Contract Design Standards or other forms of standards pertaining to the definition of service information or the structure of data exchanged by services.

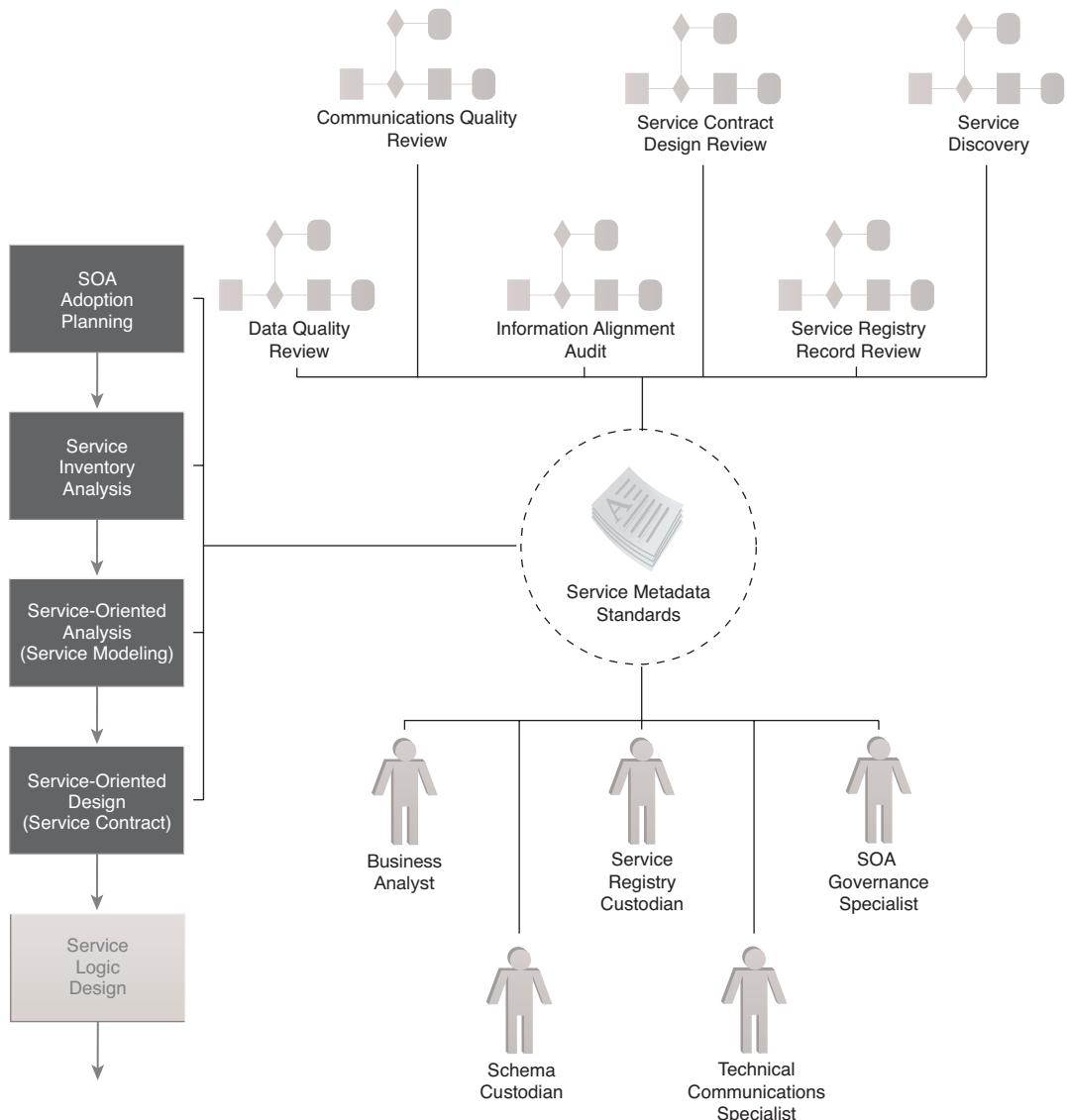
Common examples of service metadata include:

- keywords used within message headers, service contract documents, service registries, and SLAs
- elements and attribute names used by different XML schemas (including canonical XML schemas)

As with establishing a centralized business dictionary, creating an official collection of service metadata will often be limited in scope to a particular service inventory. Further, service metadata items commonly can (and should) be mapped to terms defined in the overarching business dictionary.

SOA PRINCIPLES & PATTERNS

The Messaging Metadata [535] pattern represents a form of service metadata, as commonly separated into the header portion of a message. The four metadata types (Functional, Technology, Programming Logic, Quality of Service) listed earlier pertain to the Service Abstraction (478) principle, as it is applied to determine what forms of service metadata to publish and hide.

**Figure 12.4**

Service metadata governance can begin with the SOA Adoption Planning stage and can continue throughout the Service Inventory Analysis and Service-Oriented Analysis stages. However, it is within the Service-Oriented Design stage where the actual metadata is commonly incorporated and when this precept needs to be fully applied to ensure standardization across services.

Related Processes

- Data Quality Review
- Communications Quality Review
- Information Alignment Audit
- Service Contract Design Review (Chapter 9)
- Service Registry Record Review (Chapter 11)
- Service Discovery (Chapter 11)

Related Roles

- Business Analyst
- Schema Custodian
- Service Registry Custodian
- Technical Communications Specialist
- SOA Governance Specialist

NOTE

Service registry and/or repository products are often used to store service metadata information. These types of SOA governance technologies are described in Chapter 14.

For more information regarding the aforementioned metadata types, see Chapter 8 in the *SOA Principles of Service Design* book.

Enterprise Ontology/Domain Ontology

An ontology allows the semantic relationship between two pieces of information to be inferred. This precept requires that the information that forms the basis of an ontology originates from or relates to information in the business dictionary, as well as service metadata. The level of insight established by the relationships defined within an ontology can help provide increased depth and clarity of the data's meaning.

As with the business dictionary, enterprise and domain versions of the precept for establishing an ontology exist. The same considerations explained in relation to business dictionary scope apply even more so to the definition of an ontology. For larger IT

enterprises, the creation of a proper Enterprise Ontology can be an exhaustive analysis exercise because of the multitude of relationships that need to be mapped and considered. The Domain Ontology precept is therefore more commonly applied, even when the Enterprise Business Dictionary has been established.

In cases where an ontology already exists within the IT enterprise, this precept is concerned with ensuring that the ontology is current and in alignment with business dictionary and service metadata content.

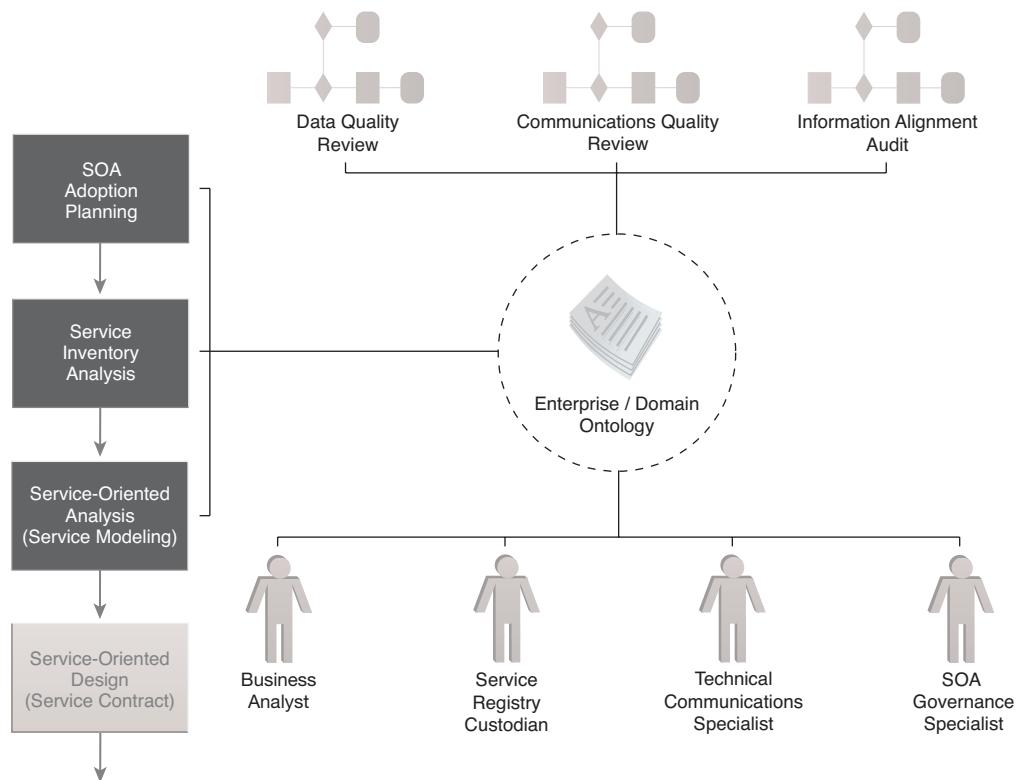


Figure 12.5

When an ontology is available, this precept is applied to the same stages as the Enterprise Business Dictionary/Domain Business Dictionary precept, namely SOA Adoption Planning, Service Inventory Analysis, and Service-Oriented Analysis.

NOTE

An ontology can be created using industry standards, such as the Web Ontology Language (OWL) and Resource Description Framework (RDF). Both of these standards allow for the definition of machine-readable concepts and relationships. Further, the Open Group developed the SOA Ontology standard to help bridge the gap between common business and IT concepts and terminology.

Related Processes

- Data Quality Review
- Communications Quality Review
- Information Alignment Audit
- Service Contract Design Review (Chapter 9)

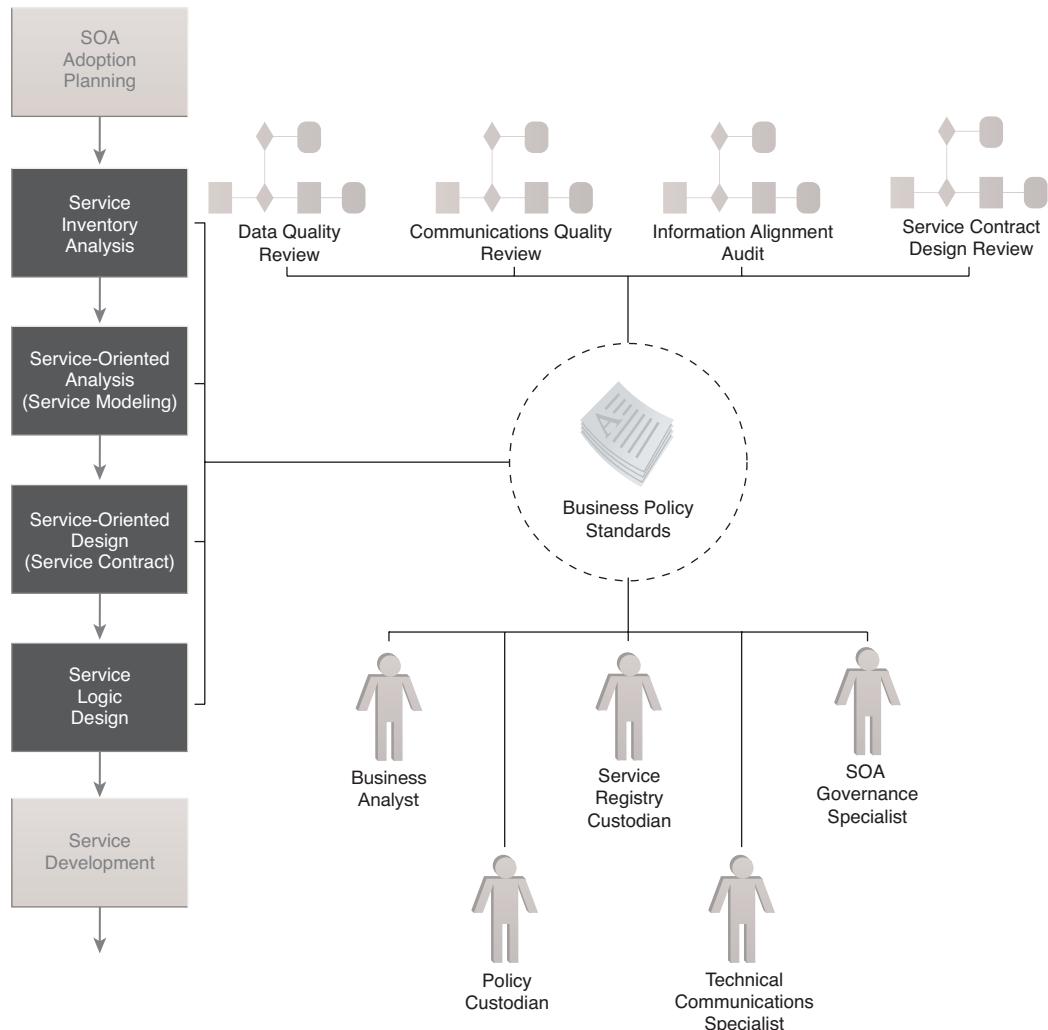
Related Roles

- Business Analyst
- Service Registry Custodian
- Technical Communications Specialist
- SOA Governance Specialist

Business Policy Standards

A business policy describes a certain aspect of how business is conducted. It can be a business rule or objective. Although commonly expressed in human-readable format, tools exist that allow for the definition of business policy content via machine-readable formats.

A common challenge to establishing technical business policies is the requirement to create and standardize a technical business policy vocabulary, proprietary to the organization. Although such a vocabulary can be derived from an existing business dictionary, making it machine-readable requires that its usage is standardized as part of precepts associated with the Service-Oriented Design project stage (and perhaps also the Service Logic Design stage for when embedded policy logic exists).

**Figure 12.6**

The standardization of business policies can become a concern during initial analysis stages, but then becomes an important requirement when technical policies are physically defined during the Service-Oriented Design stage or, if necessary, during the actual Service Logic Design stage.

This precept does not mandate the use of policies. It requires that any policies (human- or machine-readable) used within a given service inventory be standardized in terms of technology, tooling, and vocabulary.

Related Processes

- Data Quality Review
- Communications Quality Review
- Information Alignment Audit
- Service Contract Design Review (Chapter 9)

Related Roles

- Business Analyst
- Policy Custodian
- Service Registry Custodian
- Technical Communications Specialist
- SOA Governance Specialist

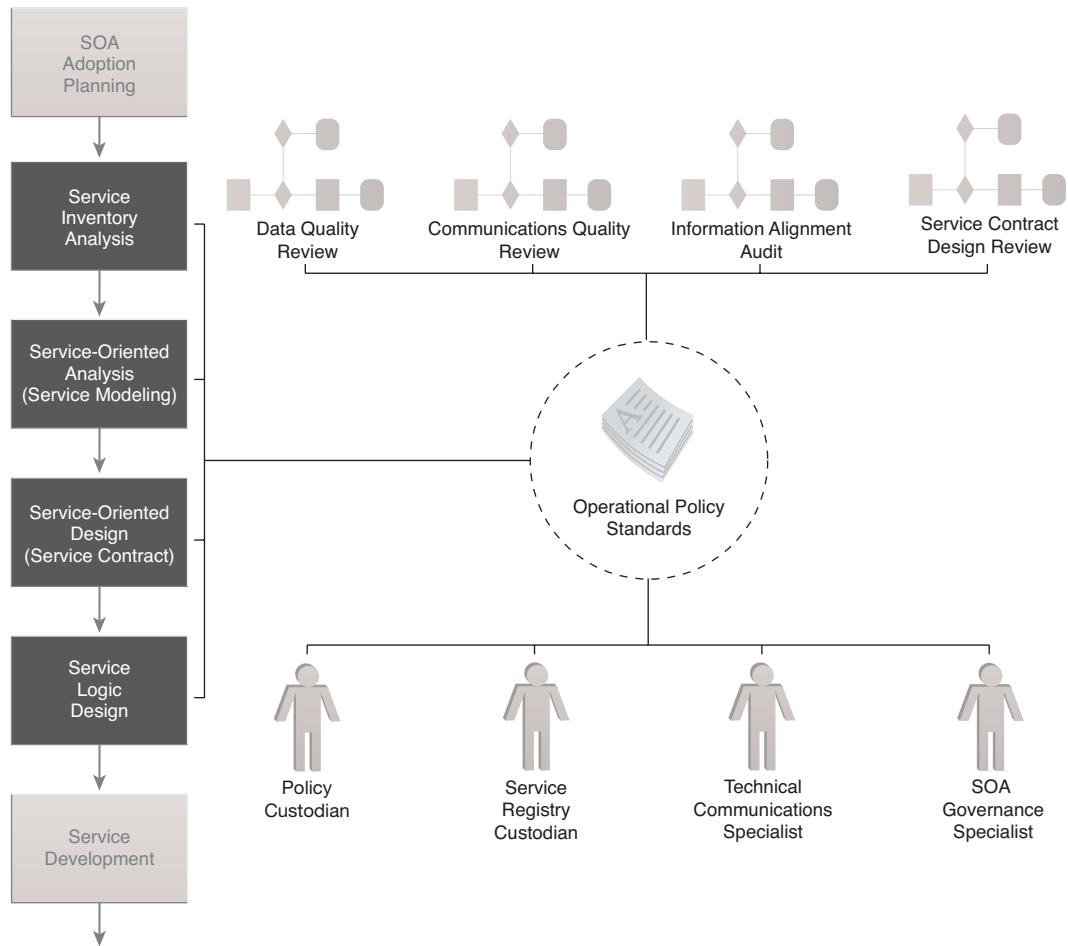
Operational Policy Standards

An operational policy (also referred to as a utility policy) can provide rules and guidelines that establish constraints and requirements for how services operate and interoperate at runtime. These types of policies are utility-centric, meaning that they influence the handling of operational characteristics that are not implemented within business policies.

Common types of processing for which operational policies are used include:

- security
- logging
- messaging protocol
- versioning

While any of these policies can be expressed in human-readable format, operational policies are almost always represented in machine-readable form. As with technical business policies, operational policies can be defined via custom vocabularies. However, the WS-Policy standard, together with various associated industry standards, provide a range of pre-defined sets of XML vocabularies for common functional areas.

**Figure 12.7**

Because operational policies can be human-readable and machine-readable (as with business policies), the same project stage associations apply, namely Service Inventory Analysis, Service-Oriented Analysis, Service-Oriented Design, and Service Logic Design.

As with the Business Policy Standards precept, this precept does not mandate the usage of operational policies. It only dictates that policies that are in use are standardized within the service inventory boundary.

Related Processes

- Data Quality Review
- Communications Quality Review

- Information Alignment Audit
- Service Contract Design Review (Chapter 9)

Related Roles

- Policy Custodian
- Service Registry Custodian
- Technical Communications Specialist
- SOA Governance Specialist

Policy Centralization

More often than not, a technical policy will apply to more than one service. In this case, the policy definition needs to be centralized so that it can be shared by all affected service contracts (Figure 12.8).

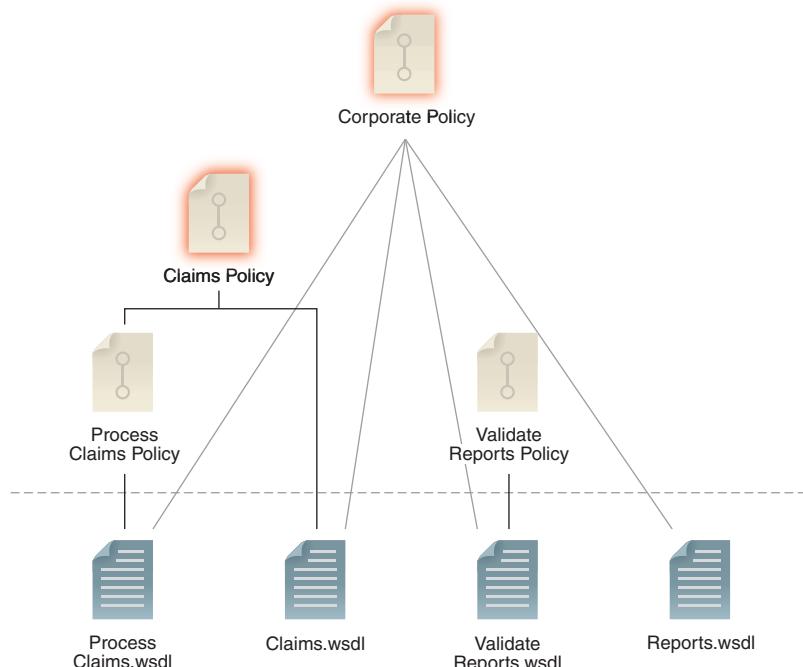
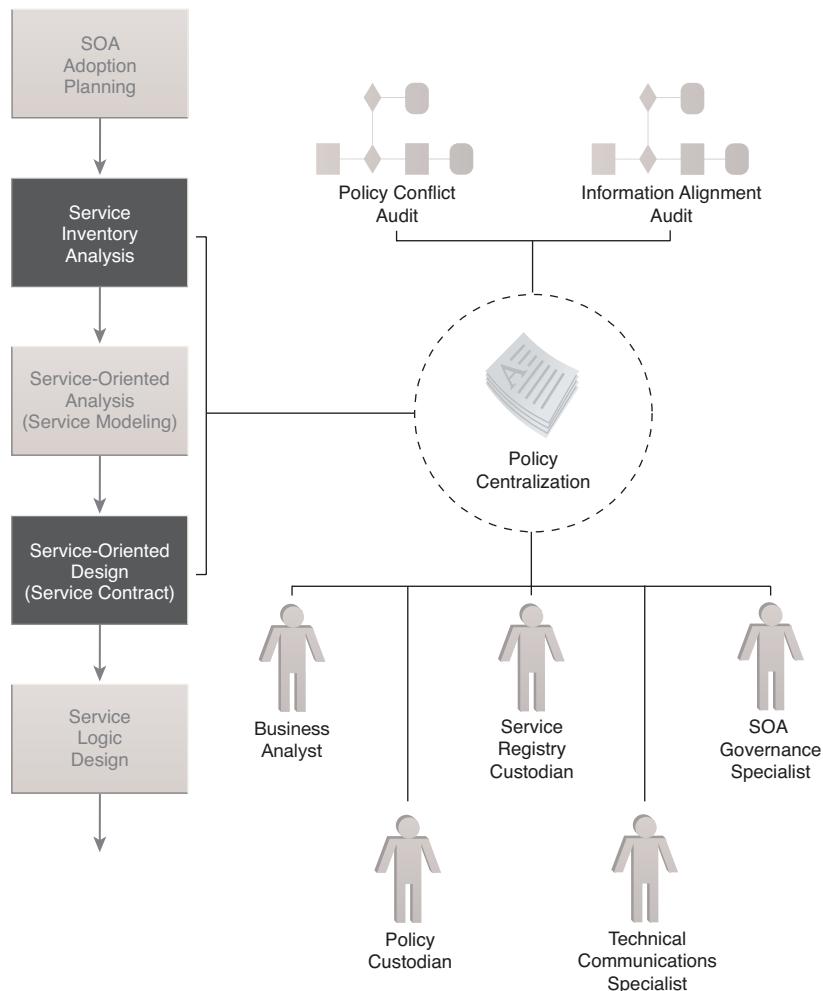


Figure 12.8

The highlighted policy documents in this diagram are being shared by multiple Web service contracts. (This figure is taken from the Policy Centralization [543] pattern description in the *SOA Design Patterns* book.)

**Figure 12.9**

This precept is concerned with centralizing policy content for use by multiple services and therefore becomes applicable during the Service Inventory Analysis stage and then the actual Service-Oriented Design stage.

By applying the Policy Centralization precept, we effectively establish logical policy domains (or layers). For business policies these policy domains are commonly aligned with line of business domains, whereas operational policies often establish independent domains that may reside within or overlap business policy domains. Some policies are considered global in that they simply apply to all services within the service inventory.

SOA PRINCIPLES & PATTERNS

The Policy Centralization precept is directly based upon the Policy Centralization [543] pattern.

Related Processes

- Information Alignment Audit
- Policy Conflict Audit

Related Roles

- Business Analyst
- Policy Custodian
- Service Registry Custodian
- Technical Communications Specialist
- SOA Governance Specialist

SUMMARY OF KEY POINTS

- Governance precepts that pertain to service information focus on the definition of a business dictionary, ontology, and business-centric metadata.
 - Governance precepts that pertain to service policies focus on the standardization of business and operational policies, as well as the centralization of policies within a service inventory.
-

Processes

The following governance processes support the application of the preceding precepts and further provide controls for maintaining the quality and alignment of service information and policies.

Data Quality Review

As previously established, information represents data with meaning. However, for us to assign the correct meaning to data, the data itself must be correct. The purpose of the Data Quality Review process is to ensure data correctness.

To address quality concerns we must first understand the characteristics and dimensions of data quality most important to the organization. Common areas of data quality that can be incorporated into this review include:

- completeness
- accuracy
- consistency
- timeliness
- relevance

When assessing data quality and undertaking quality improvement programs it is necessary to identify and prioritize these characteristics according to business requirements. SOA Quality Assurance Specialists can then establish metrics used to track quality improvement over time.

Once key data quality characteristics and associated metrics have been identified, data profiling can be conducted against existing data sources to assess current-state quality. Subsequently, data cleansing and “de-duplication” efforts can be carried out to further improve the quality of existing data. Several vendors provide supporting tools for these tasks.

Related Precepts

- Enterprise Business Dictionary / Domain Business Dictionary
- Service Metadata Standards
- Enterprise Ontology / Domain Ontology
- Business Policy Standards
- Operational Policy Standards

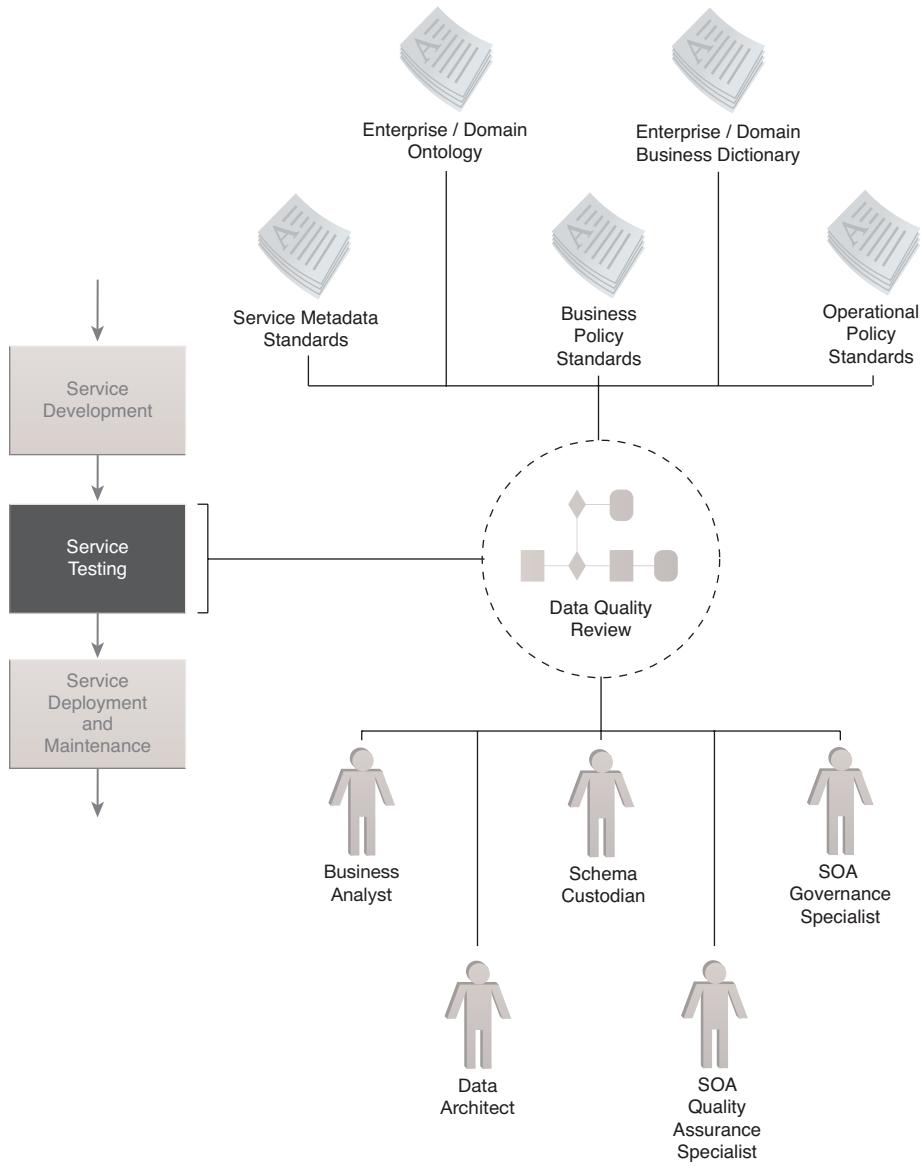


Figure 12.10

The Data Quality Review is generally considered a quality assurance process and is therefore commonly carried out as part of the Service Testing stage.

Related Roles

- Business Analyst
- Data Architect
- Schema Custodian
- SOA Quality Assurance Specialist
- SOA Governance Specialist

Communications Quality Review

While the Data Quality Review is focused on the correctness quality of data, it does not typically address the communications quality of data. Communications quality is the primary concern of the Technical Communications Specialist role and is focused on making published data (and information) interpretable and discoverable.

This review can be carried out for any body of content published as a result of the aforementioned precepts, regardless of whether the deliverable artifacts are human-readable or machine-readable.

SOA PRINCIPLES & PATTERNS

Depending on the nature of the data, this precept can relate to the application of the Service Discoverability (484) principle, which requires that service metadata be authored for discoverability and interpretability by a wide range of project team members.

Related Precepts

- Enterprise Business Dictionary / Domain Business Dictionary
- Service Metadata Standards
- Enterprise Ontology / Domain Ontology
- Business Policy Standards
- Operational Policy Standards

Related Roles

- Business Analyst
- Technical Communications Specialist
- SOA Quality Assurance Specialist
- SOA Governance Specialist

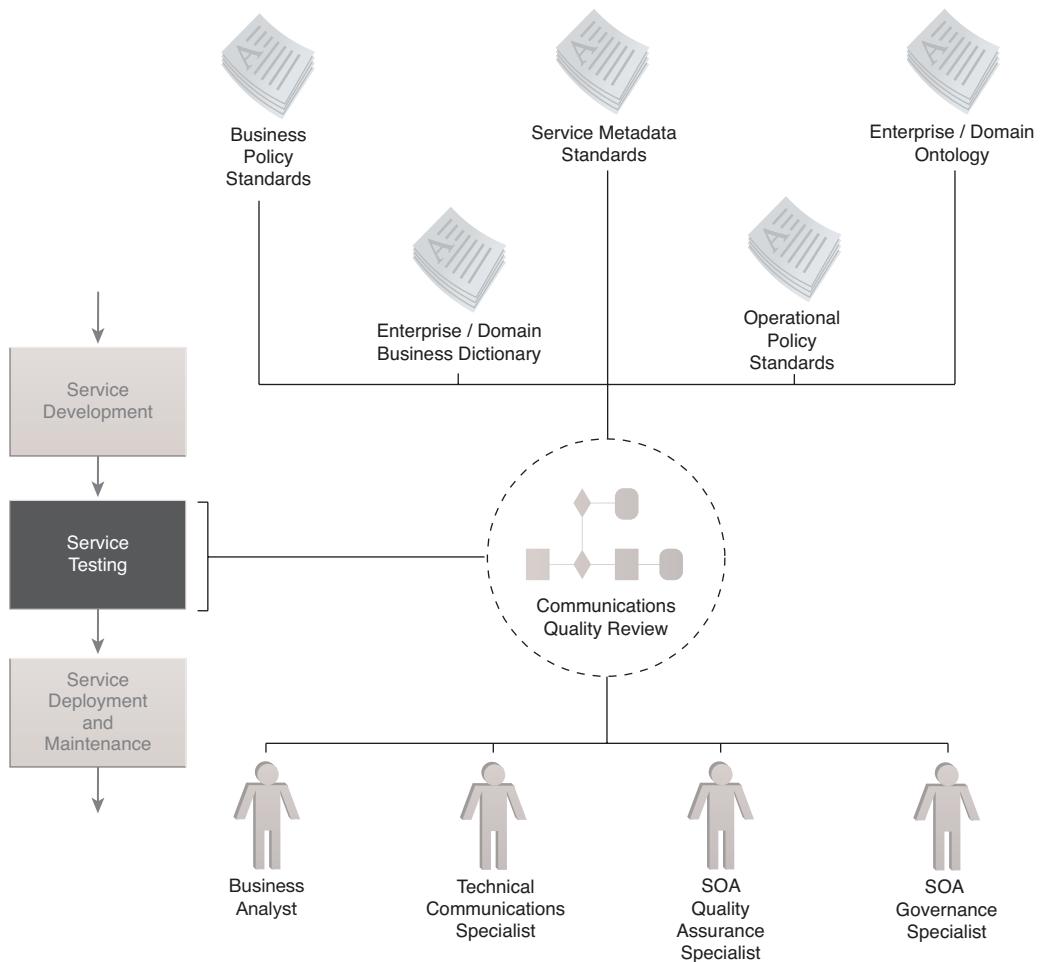


Figure 12.11

Also classified as a quality assurance process, the Communications Quality Review is commonly performed during the Service Testing stage.

Information Alignment Audit

When establishing bodies of content that define information (give data specific meaning) or bodies of content that then use or build upon previously defined information, we need to ensure that the meaning is kept consistent throughout the scope in which all bodies of content are utilized.

This process essentially validates the alignment of information across different bodies of content, such as the previously described business dictionary, metadata, policies, and ontology. It involves steps that trace one piece of data or information back to its origin where meaning was first assigned.

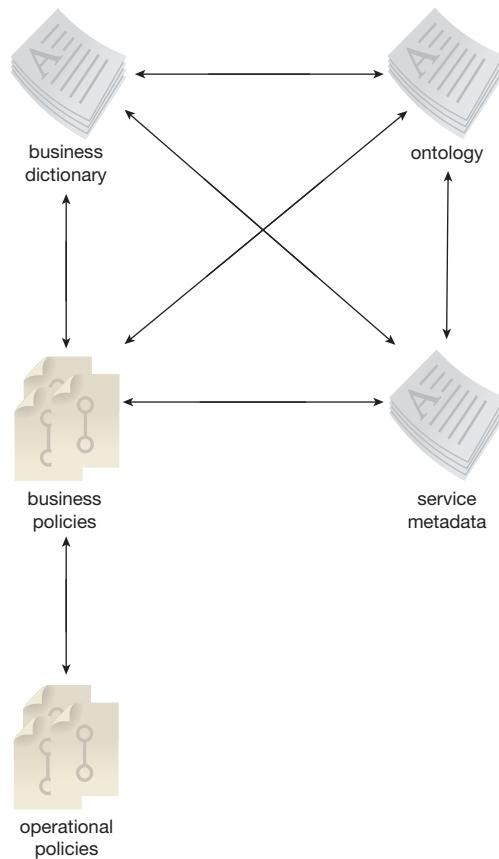


Figure 12.12

An example of how different bodies of business intelligence can relate to and depend upon each other. The Information Alignment Audit needs to trace and validate all of these relationships and dependencies.

Typically, an Information Alignment Audit is carried out iteratively, for each newly delivered or expanded service or service-oriented solution that introduces relevant business data or information.

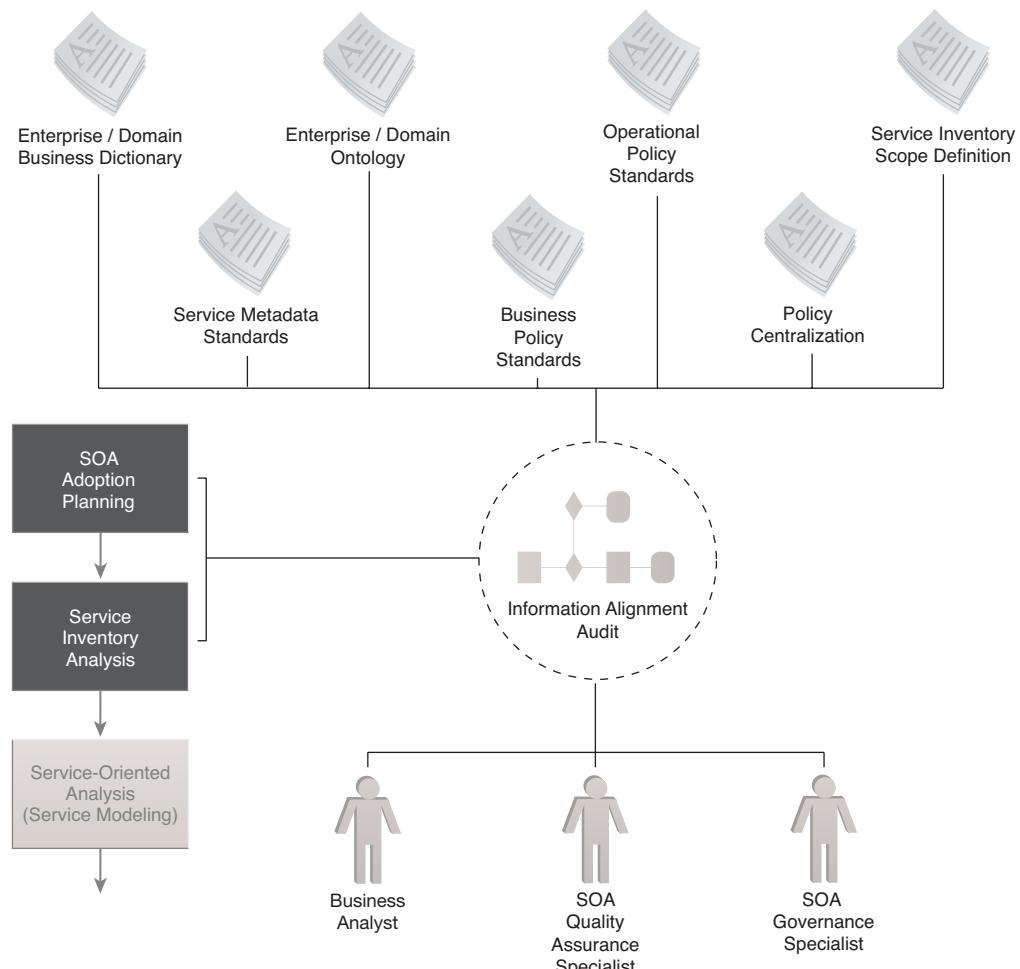


Figure 12.13

Tracing the alignment between different bodies of business information needs to be performed as early in the project lifecycle as possible, before services are built to actually incorporate and use the data. As a result, this process is commonly carried out during the initial SOA Project Planning and Service Inventory Analysis stages. Although the governance of several precepts further extend into Service-Oriented Analysis, cross-artifact alignment concerns should have been resolved prior to that stage.

Related Precepts

- Enterprise Business Dictionary / Domain Business Dictionary
- Service Metadata Standards
- Enterprise Ontology / Domain Ontology
- Business Policy Standards
- Operational Policy Standards
- Policy Centralization
- Service Inventory Scope Definition (Chapter 7)

Related Roles

- Business Analyst
- SOA Quality Assurance Specialist
- SOA Governance Specialist

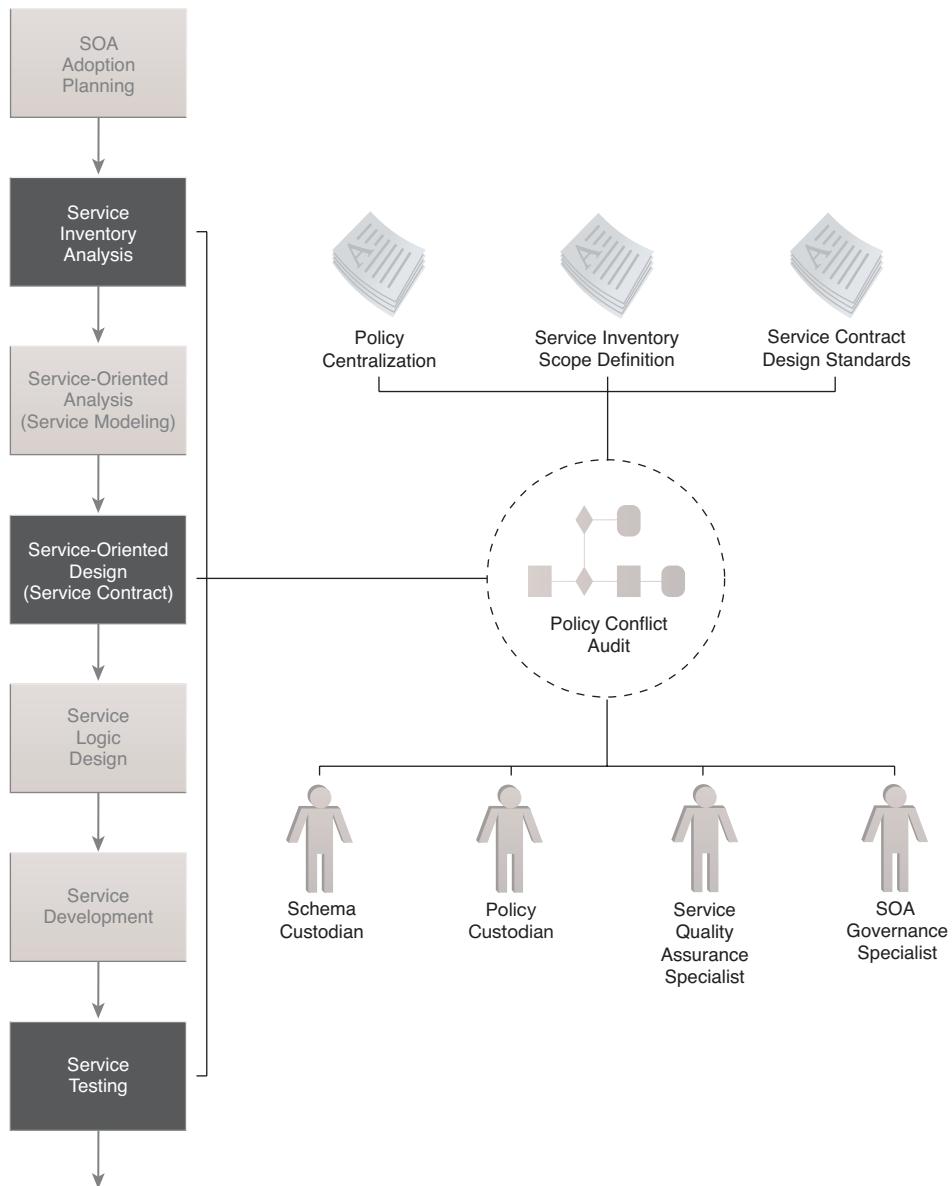
Policy Conflict Audit

When centralizing policies (as per the Policy Centralization precept and pattern), each policy establishes its own scope of application (or domain). Because one service can be associated with multiple policies, it is natural for policy domains to overlap. When defining a centralized policy, it is therefore important to take into account the nature of the constraints and rules the policy is imposing upon affected services. If a constraint or rule in a newly shared policy conflicts with a constraint or rule in an existing policy (shared by the same service), then various unpredictable runtime exceptions and behaviors can occur. The greater the domain or applicability of a policy, the greater the risk of conflicts with existing policies.

The Policy Conflict Audit process requires that, prior to introducing a new policy within a service inventory, the constraints and rules in the proposed policy are cross-checked against any affected policies (as well as other relevant service contract content that defines constraints and rules).

Related Precepts

- Policy Centralization
- Service Inventory Scope Definition (Chapter 7)
- Service Contract Design Standards (Chapter 9)

**Figure 12.14**

Policy conflicts are ideally resolved before policy definitions are even introduced to the service contract and service architecture. This process can therefore be part of the Service Inventory Analysis, Service-Oriented Design, or Service Testing stages.

Related Roles

- Schema Custodian
- Policy Custodian
- SOA Quality Assurance Specialist
- SOA Governance Specialist

SUMMARY OF KEY POINTS

- Service information governance processes are primarily concerned with the quality assurance of business information relevant to services.
 - Policy conflict resolution is a key factor for successfully using shared policies within service inventories.
 - All of the artifacts resulting from the application of service information and policy precepts need to be kept in alignment in order to maintain their value.
-

People (Roles)

The availability of service information and policy deliverables and artifacts can end up affecting just about any organizational role within an SOA project. Service Analysts in particular are required to work with business-centric and human-readable deliverables when identifying and defining service candidates during early analysis stages. However, our focus is on roles associated with the *governance* of service information and policies. This brings us back to the previously described precepts and processes and the people commonly involved in their definition and execution.

Business Analyst

Business Analysts and other types of business subject experts are generally the most qualified people to contribute to the authoring of a business dictionary and related ontology. Their insight into how lines of business and related business processes have and continue to operate gives them a clear understanding of how different business entities exist, how they relate, and how they should be represented by terms and definitions. This same business-centric expertise can also be valuable for the identification and definition of business-centric metadata and policies, as well as participation in processes that involve the review of business information.

Related Precepts

- Enterprise Business Dictionary/Domain Business Dictionary
- Service Metadata Standards
- Enterprise Ontology/Domain Ontology
- Business Policy Standards
- Policy Centralization

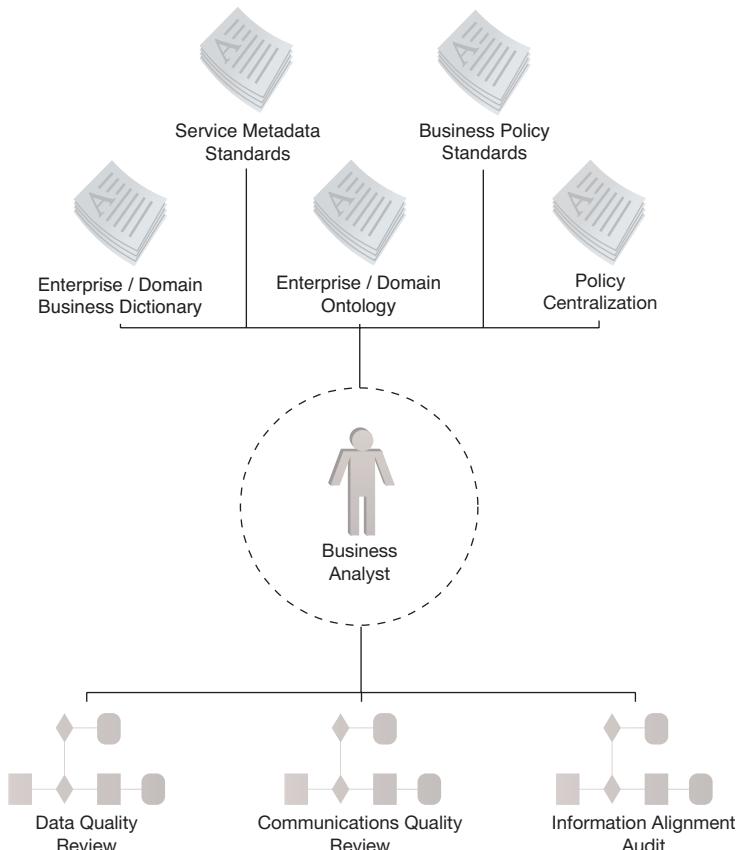


Figure 12.15

Service Information and Service Policy governance precepts and processes associated with the Business Analyst role.

Related Processes

- Data Quality Review
- Communications Quality Review
- Information Alignment Audit

Data Architect

Because of their intimate knowledge of data sources and how data and data relationships, at a physical storage level, have been modeled, Data Architects are generally the primary participants of the Data Quality Review process.

Related Precepts

N/A

Related Processes

- Data Quality Review

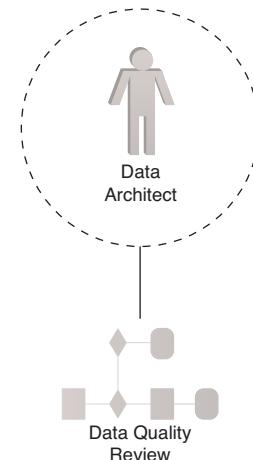


Figure 12.16

Service information and service policy governance precepts and processes associated with the Data Architect role.

Schema Custodian

XML schemas, and other forms of data models that may be used or referenced by service contracts and message structures, are often derived from (or required to encapsulate) underlying data sources and architectures. Schema Custodians have expertise in the definition of schemas (and often also the definition of schema design standards). This expertise can be helpful or even required for the identification and definition of suitable service metadata. Further, Schema Custodians can assist with Data Quality Reviews and may be asked to help resolve issues raised during Policy Conflict Audits.

Related Precepts

- Service Metadata Standards

Related Processes

- Data Quality Review
- Policy Conflict Audit

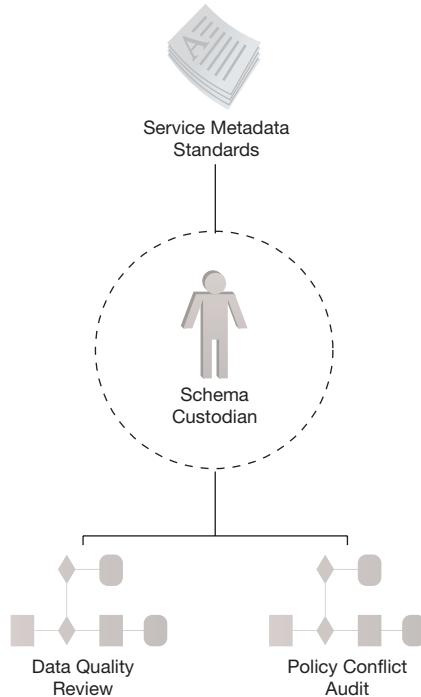


Figure 12.17

Service information and service policy governance precepts and processes associated with the Schema Custodian role.

Policy Custodian

The Policy Custodian role is most commonly associated with the definition and maintenance of machine-readable policies. However, part of this responsibility often requires the interpretation (and, in some cases, the initial definition) of parent human-readable policies. Either way, this role is directly involved with all policy-related precepts and processes.

Related Precepts

- Business Policy Standards
- Operational Policy Standards
- Policy Centralization

Related Processes

- Policy Conflict Audit

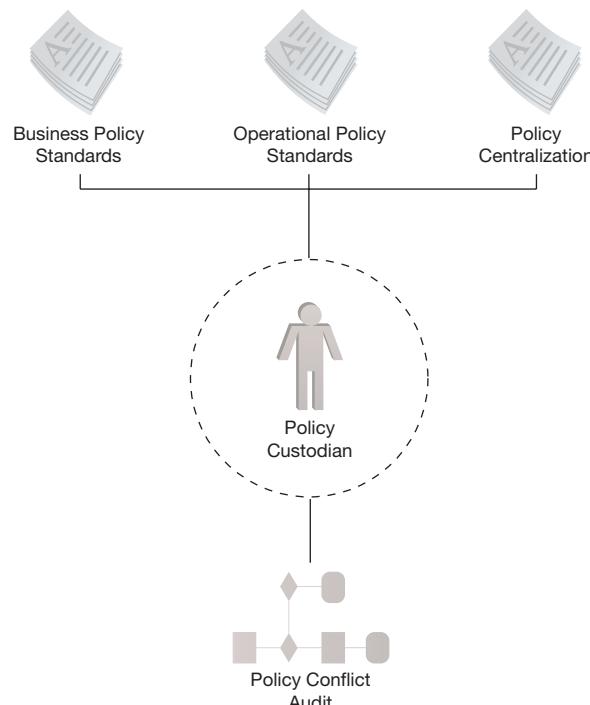


Figure 12.18

Service information and service policy governance precepts and processes associated with the Policy Custodian role.

Service Registry Custodian

Any of the bodies of content resulting from service information and policy-related precepts can be stored or represented within a central service registry. In this case, the Service Registry Custodian can contribute to the governance tasks pertaining to the application of the precepts and the delivery of the resulting information and policy artifacts. For example, the chosen service registry product may have certain features or constraints that end up shaping the format of business data or the manner in which meaning is associated with the data.

Related Precepts

- Enterprise Business Dictionary / Domain Business Dictionary
- Service Metadata Standards
- Enterprise Ontology / Domain Ontology

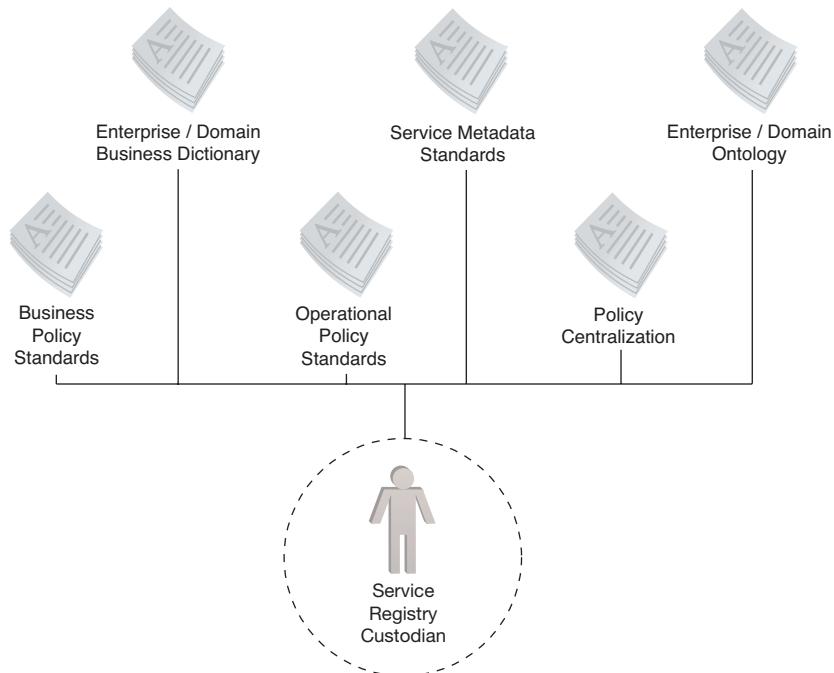


Figure 12.19

Service information and service policy governance precepts and processes associated with the Service Registry Custodian role.

- Business Policy Standards
- Operational Policy Standards
- Policy Centralization

Related Processes

N/A

Technical Communications Specialist

Precepts that govern the delivery of bodies of content that are published in human-readable and machine-readable format will usually make the quality of such content a priority. The Technical Communications Specialist's responsibility is to incorporate governance controls that help regulate published content in support of ensuring interpretability and discoverability for a range of project team members.

To improve the quality of interpretation, the Technical Communications Specialist will often need to establish review and revision cycles that enable content to undergo required wording and vocabulary changes. To increase discoverability, the expression of content will sometimes need to be simplified. Although the Technical Communications Specialist will be involved in the development of the content, this role may also be part of the actual review of the content's communications quality.

Related Precepts

- Enterprise Business Dictionary / Domain Business Dictionary
- Service Metadata Standards
- Enterprise Ontology / Domain Ontology
- Business Policy Standards
- Operational Policy Standards

Related Processes

- Communications Quality Review

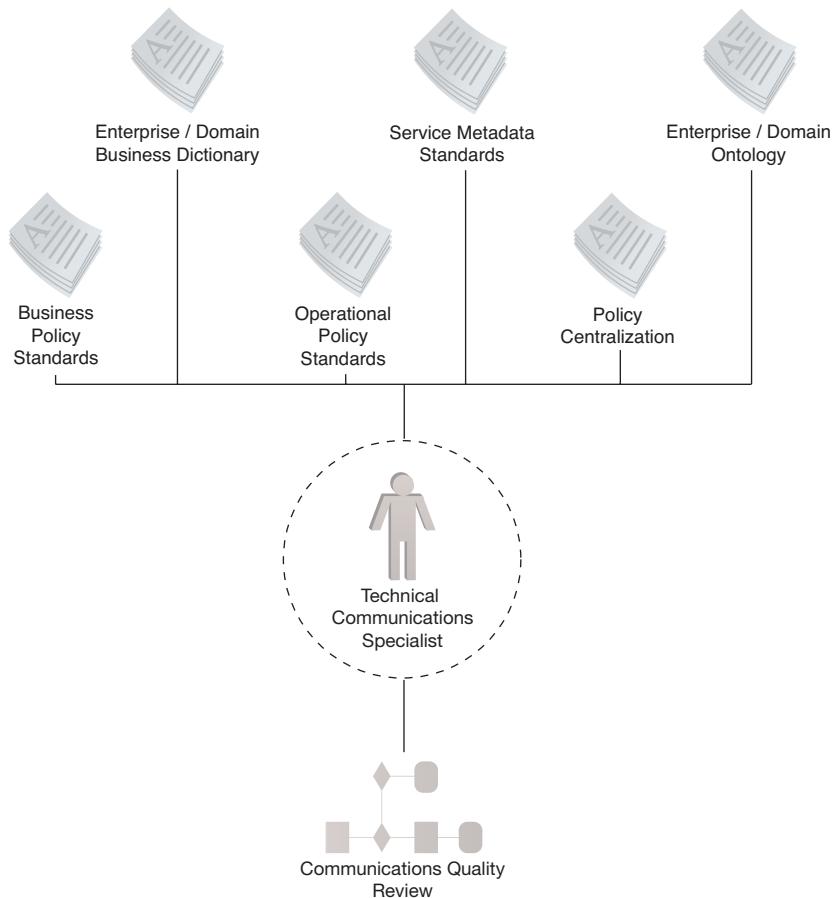


Figure 12.20

Service information and service policy governance precepts and processes associated with the Technical Communications Specialist role.

SOA Quality Assurance Specialist

The participation of this role is vital for the Data Quality Review and Policy Conflict Audit processes, as well as the Communications Quality Review process. It also makes sense to have an SOA Quality Assurance Specialist act in an advisory capacity for the governance of any of the service information and policy processes.

Specifically, these professionals can be involved in the following areas:

- ensuring that relevant quality assurance practices and metrics are applied
- incorporating processes within (and relating the processes to) quality assurance methodologies and processes already being used

Basically, whenever any of the service information and service policy precepts can be associated with the Service Testing stage, an SOA Quality Assurance Specialist should participate to ensure proper positioning of the precepts within the overall testing methodology.

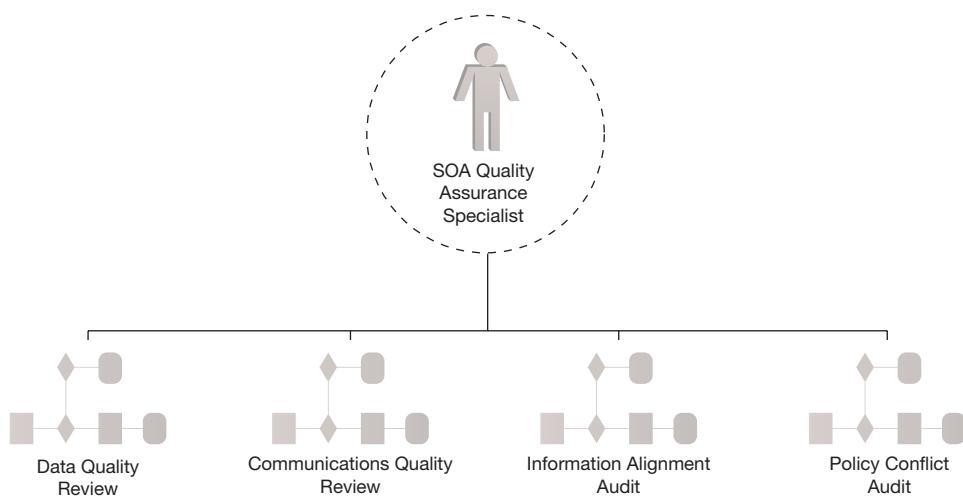


Figure 12.21

Service information and service policy governance precepts and processes associated with the SOA Quality Assurance Specialist role.

Related Precepts

N/A

Related Processes

- Data Quality Review
- Communications Quality Review
- Information Alignment Audit
- Policy Conflict Audit

SOA Governance Specialist

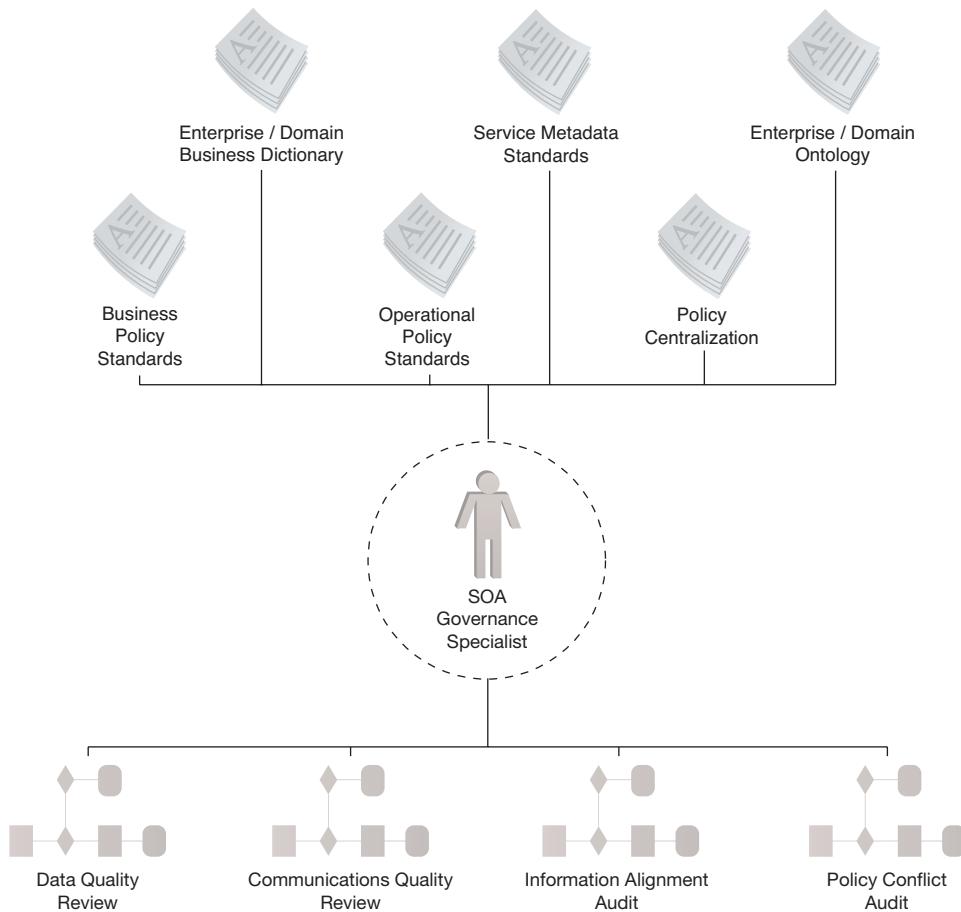
The SOA Governance Specialist is required to be involved in the identification, definition, and governance of all service information and policy precepts and processes. As members and representatives of the SOA Governance Program Office, individuals fulfilling this role will be required to supervise and, in some cases, lead efforts required to establish these precepts and processes.

Related Precepts

- Enterprise Business Dictionary/Domain Business Dictionary
- Service Metadata Standards
- Enterprise Ontology/Domain Ontology
- Business Policy Standards
- Operational Policy Standards
- Policy Centralization

Related Processes

- Data Quality Review
- Communications Quality Review
- Information Alignment Audit
- Policy Conflict Audit

**Figure 12.22**

Service information and service policy governance precepts and processes associated with the SOA Governance Specialist role.

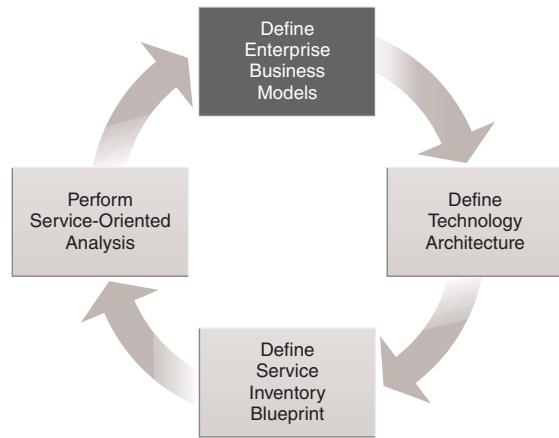
SUMMARY OF KEY POINTS

- Multiple organizational roles can be involved with any one service information or policy precept or process.
- Especially for business information governance, it can be necessary to draw upon the expertise of several business subject matter experts.

12.3 Guidelines for Establishing Enterprise Business Models

Several of the artifacts referenced in the preceding *Precepts* section of this chapter represent foundational business intelligence used as input for the Service Inventory Analysis and Service-Oriented Analysis project stages. These are collectively referred to as *enterprise business models*.

The *Define Enterprise Business Models* step in the Service Inventory Analysis lifecycle is dedicated to ensuring that the necessary business artifacts are in place and sufficiently up-to-date. All of this business content can be considered business information (business data with meaning). The following guidelines provide recommendations for establishing and maintaining enterprise business models in support of the overall SOA initiative.



Establish a Service Information Governance Council

Because business information spans organizational boundaries, it can be helpful (and sometimes necessary) to form a council (or steering committee) comprised of members or department representatives from all affected lines of business. This type of council would typically be joined by senior members of management with high-level decision-making authority. It is positioned as a sub-office or sub-group of the SOA Governance Program Office and receives its mandate from that office, as well as executive sponsors. If a general Business Information Council or governing body also exists, then it would need to be aligned (or also become a parent entity to) the Service Information Governance Council.

Assign Business Information Custodians

A role not defined in this book is the Business Information Custodian. Although it is not specific to SOA projects, it can be a valuable role to establish in support of creating and keeping current a business dictionary and ontology. This form of custodianship responsibility can be assigned to multiple individuals, each with specific business subject matter expertise. They would typically receive authority and direction from the Service Information Governance Council.

When available, Business Information Custodians can be effective contributors to the Information Alignment Audit process.

Assign Value to Business Information

Some forms of business information are core and critical to an organization's primary business tasks, while others may be peripheral or supplementary in nature. When documenting business information in the form of a business dictionary, it can be helpful to assign individual terms and business entities a value ranking indicating its relative importance to the organization. These ranks can help Business Analysts, Service Analysts, and SOA Governance Specialists prioritize the definition, maintenance, and incorporation of business information within SOA project lifecycle stages. This consideration is especially relevant when a project delivery methodology is being employed whereby only limited time and effort is being assigned to up-front analysis stages.

Relate Service Information Governance to Master Data Management

Master data management (MDM) is an accepted practice for governing reference data that is deemed core to an organization's business. It has become a data governance-related discipline with its own well-defined roles, responsibilities, processes, and technology products. As a result, if MDM is in use within an IT enterprise that is adopting SOA, it should be related to and incorporated with service information and policy governance precepts and processes.

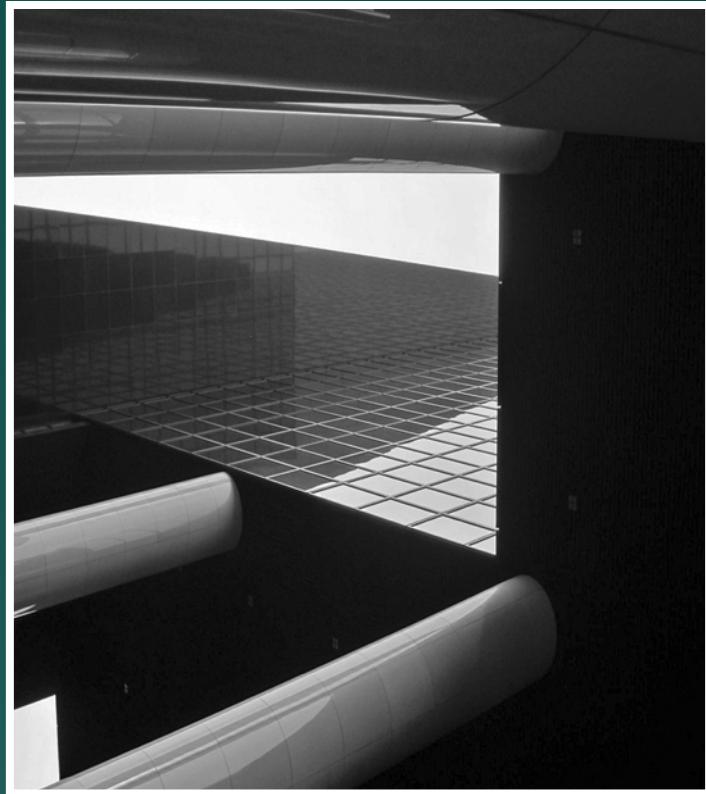
For example, MDM data domains are usually classified according to fundamental business entities common to an entire organization (such as customer, product, location, supplier, etc.). Given that this same data and information can form the basis of business entity services that are commonly modeled as part of Service-Oriented Analysis processes, advance planning will help ensure that MDM and SOA governance systems are complementary.

SUMMARY OF KEY POINTS

- Business enterprise models encompass several of the service information artifacts relevant to SOA governance and SOA projects in general.
 - The usage of business enterprise models, such as the business dictionary and ontology, can go well beyond the scope of an SOA project. Therefore additional practices can apply.
-

This page intentionally left blank

Chapter 13



SOA Governance Vitality

13.1 Vitality Fundamentals

13.2 Vitality Triggers

13.3 SOA Governance Vitality Process

Investments made into SOA initiatives need to be viewed as living assets. Once deployed and in use, services, service-oriented solutions, and the technology architectures and infrastructure resources that support and realize their runtime enablement need to continue to fulfill the expectations and requirements of the business. The SOA governance program must therefore provide a means by which these assets are routinely reviewed, kept current and accurate and, most importantly, relevant to (often changing) business needs.

To address this evolutionary aspect, the SOA governance program needs to make the notion of *vitality* part of the SOA governance system being employed. *SOA governance vitality* is a proactive approach for maintaining the validity and applicability of an SOA governance system and the many artifacts, assets, and people it oversees.

SOA governance vitality typically exists as a sub-framework that is part of the overall SOA governance system. This chapter introduces common parts of this framework that pertain to on-premise and cloud-based assets.

13.1 Vitality Fundamentals

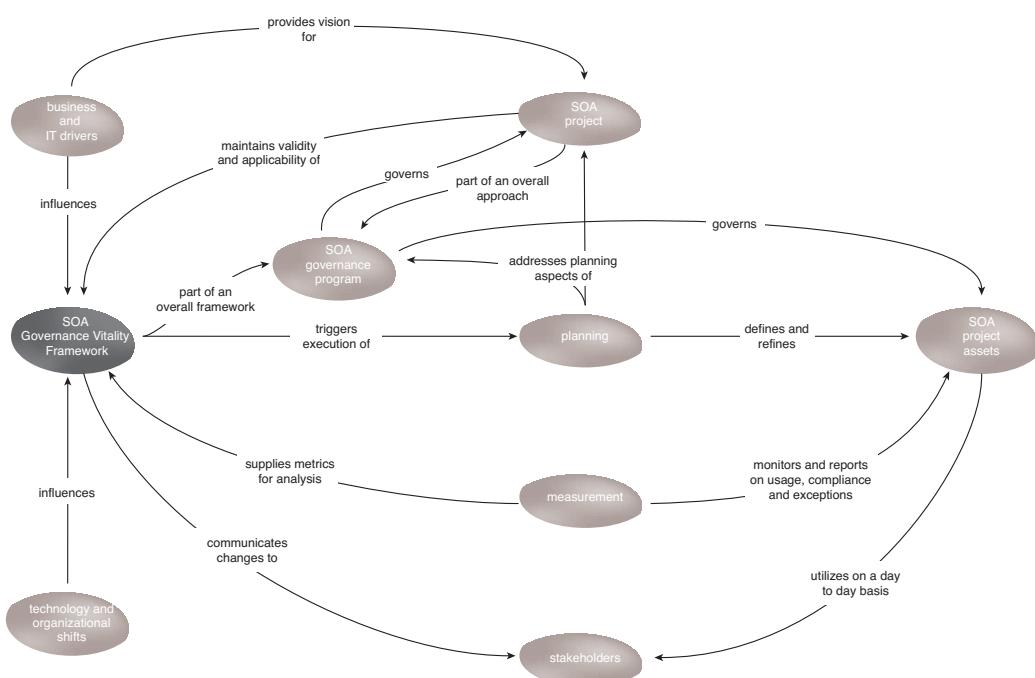
Many IT enterprises have traditionally applied governance reactively when confronted with change. The essence of service-orientation is to create an IT environment inherently capable of accommodating change. Therefore, an SOA governance program must establish a system of governance that is fully prepared for business change and that will react to this change with unwavering support for the strategic goals of the business.

An SOA governance vitality framework utilizes *vitality triggers* as a mechanism of notification and reaction. An executed trigger can result in one or more predefined activities to address the needs of whatever change “pulled the trigger.” Common forms of triggers and resulting *vitality activities* are described in the upcoming sections.

NOTE

SOA governance vitality is sometimes classified as a subset of an overall approach to SOA vitality. SOA vitality, as a topic area, can be broader. It can deal with the vitality of the overall initiative, starting with early delivery stages to post-implementation stages. As previously explained, our coverage of SOA *governance* vitality is focused on maintaining the relevancy and accuracy of the existing SOA governance system, its precepts and processes. As such, it is a framework that, when successfully applied, is commonly associated with maintaining the Business Driven level of organizational maturity (as first explained in Chapter 4).

Figure 13.1 provides an overview of how an SOA governance vitality framework can connect with other parts of the SOA governance program and areas of the IT enterprise in general. Several of these relationships are explored in this chapter.

**Figure 13.1**

Common relationships of an SOA governance vitality framework.

SUMMARY OF KEY POINTS

- Vitality, within the context of SOA, is a quality measured by an SOA initiative's ability to maintain a desired direction and attain desired goals in the face of on-going change.
- SOA governance vitality is a subset of SOA governance that is focused on extending SOA governance practices to the governance of an SOA initiative's vitality.
- An SOA governance vitality framework commonly exists as a part of the overall SOA governance system. This framework is typically comprised of one or more processes comprised of pre-defined vitality activities executed by pre-defined vitality triggers.

13.2 Vitality Triggers

Vitality triggers represent pre-determined events and conditions that execute pre-determined activities as part of a vitality process. It is important to enable a wide range of project team members to initiate certain triggers so that a broad variety of concerns can be encompassed by the SOA governance vitality framework. Figure 13.2 shows the five main categories of vitality triggers, each of which is described in the upcoming sections.

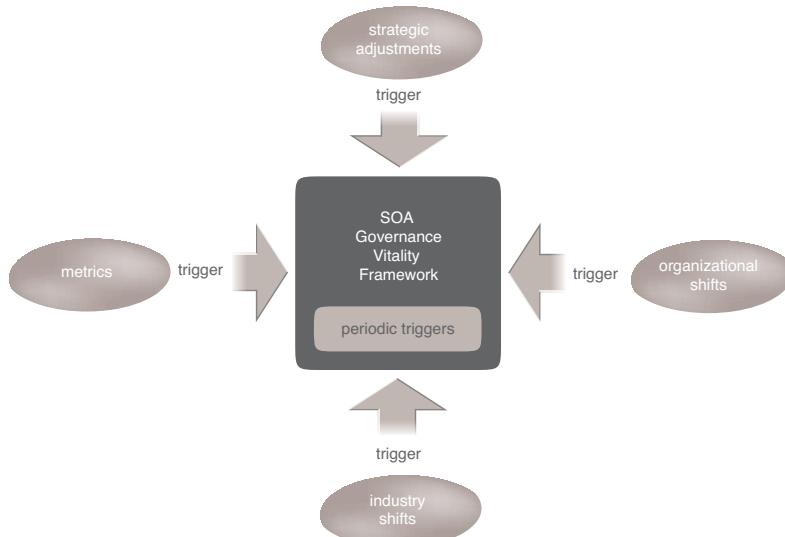


Figure 13.2

Common types of vitality triggers that can execute an SOA governance vitality process.

Vitality triggers need to be balanced in terms of importance and impact. Imbalanced vitality triggers can impose an ineffectual framework or one that unnecessarily introduces governance burden. Therefore, asset update and refresh requirements must be weighed against the importance of the trigger. The responsibility of choosing and balancing collections of vitality triggers lies with SOA Governance Specialists.

In most cases, a trigger will execute activities that result in some form of asset update or refresh. The cost and effort associated with this type of activity must be carefully considered to ensure it is justified and worthwhile. On the other hand, choosing to not trigger some activities can result in undesirable consequences to the overall SOA initiative (which can lead to new costs, effort, and other negative impacts down the road).

Business vs. Technology Changes

Vitality activities are triggered by change. Change can originate from the IT and business divisions or communities within an organization. Figure 13.3 illustrates the never-ending cycle that most organizations find themselves iterating through over time. When adopting service-orientation, the focus is mainly on business-centric requirements, which is usually the primary source of change affecting SOA initiatives. However, it is important to acknowledge and be prepared for changes that originate within the IT community, as these types of changes can, in turn, impact the business.

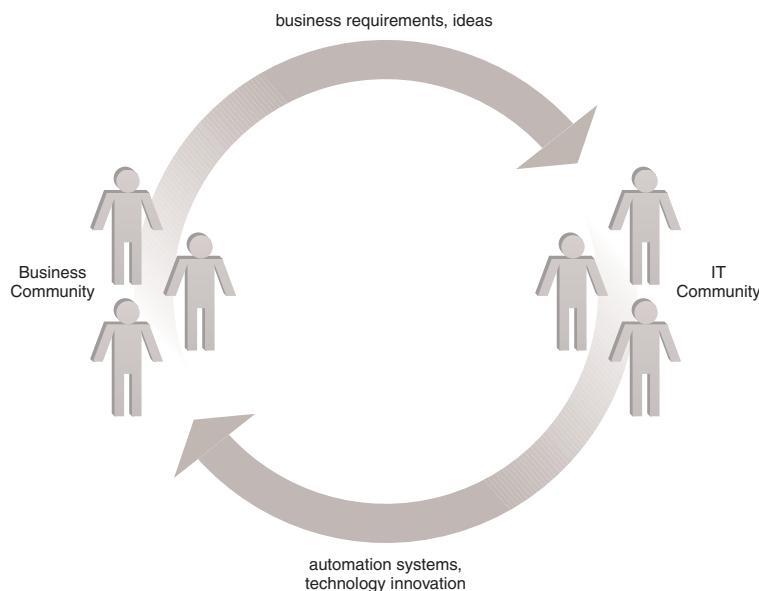


Figure 13.3

The on-going cycle of change that service-orientation is geared to inherently accommodate.

Types of Vitality Triggers

There are five common types of vitality triggers:

- Strategic Adjustments
- Industry Shifts
- Metrics
- Organizational Shifts
- Periodic

Most of these categories have further, more specific types of related triggers. The upcoming sections explain each of these triggers in more detail.

NOTE

The cycle displayed in Figure 13.3 relates specifically to the following vitality triggers:

- Strategic Business Adjustment
- Strategic IT Adjustment
- Business Shift
- Technology Shift

Other types of triggers can also fall within this cycle, depending on their nature and origin.

Strategic Adjustments

Sometimes changes are unexpected; other times they are the result of an intentional decision to change or *adjust* strategic direction. Strategic adjustments are usually significant and will almost always act as vitality triggers. Following are descriptions of business and IT strategic adjustments.

Strategic Business Adjustment

A strategic business adjustment trigger is driven by changes to the organization's business direction or vision. An example of this type of change is a corporation altering its overall business strategy as a result of a recent merger.

When business drivers are augmented to such an extent, it can reprioritize various aspects of an SOA initiative. The resulting ripple effects can be far-reaching, ranging from delivery processes to technology to the governance system itself. Therefore, this type of trigger is usually considered the most significant and potentially impactful.

Strategic IT Adjustment

IT departments can initiate changes pertaining to technology and resources (human and automated). Sometimes these changes are in support of fostering greater business requirements fulfillment (such as when new technology innovation is made available to the business) or they can be limiting and prohibitive (such as when automation requirements requested by the business cannot be fulfilled due to infrastructure, skill-sets, security, or other forms of IT limitations).

For example, an IT department traditionally limited by its budget and on-premise resources may discover that it can now begin to offer greater scalability of its automated systems to business stakeholders by leveraging cloud-based environments.

These types of adjustments to IT strategy can alter the entire complexion of a service architecture or even the service inventory architecture itself. Therefore, in some cases, regular vitality refresh activities may be insufficient.

Industry Shifts

Whereas strategic adjustment triggers generally represent situations when change is the result of deliberate decision-making by management or stakeholders, changes imposed by industry developments are generally considered triggers out of the control of the affected organization.

Business Shift

Within most public and private sectors, an organization can find itself impacted by changes in business law, governmental policy, or even economic or political factors. When external developments affect how a business can and should carry out its operations, it generally warrants the use of this vitality trigger. For example, an accounting firm may find several of its business processes heavily influenced by the release of a new tax law or subsidy.

Technology Shift

This trigger represents industry technology shifts that occur within proprietary vendor platforms and product lines, as well as industry technology standards communities.

Examples include:

- A new version of an infrastructure product becomes available and the version currently being used by the organization will no longer be supported after a period of time.
- The Service Custodian decides that a service is to be moved from an on-premise environment to a cloud-based environment (or vice versa).
- A product vendor announces that it will discontinue a product used in the IT enterprise.
- A new version of a currently utilized technology or application is now service-enabled.
- A new security industry standard is released, superseding the standard to which an enterprise's solutions currently comply.
- A cloud provider offers new leasing models for IT resources with greater amounts of scalability, or perhaps at lower rates.

These types of changes can impact various parts of technology architecture and infrastructure, as well as the skill-sets and resources required to implement and support the required technology changes.

Metrics

The most common form of vitality trigger is based on the use of metrics. An SOA governance framework can utilize quantitative and qualitative metrics to identify inadequacies, exceptions, violations, and other problem areas. Not only do metrics provide visibility into the effectiveness of an SOA governance system, they can also indicate if parts of the system are too strict, arduous, or otherwise unsuccessful.

It is important that a manageable set of metrics is created and that each metric has a clearly defined rationale. Too many metrics can be overwhelming, too few ineffectual. Initial metrics are likely to be subject to further refinement as the SOA governance vitality framework matures.

A variety of metrics can be defined in support of vitality requirements. Provided here are descriptions for two common types.

Performance Metrics

Metrics with defined thresholds can raise triggers to assist in measuring the performance of various parts of an SOA ecosystem.

Example areas of focus include:

- the operational efficiency of services or specific service capabilities
- the operational burden of a service-oriented solution in relation to its ROI
- the utilization of a given on-premise physical server compared to its purchase and administration cost
- the usage costs of a cloud-based service that is incurring per-usage fees
- the administration costs associated with resolving recurring security problems of cloud-based services being periodically attacked

It should be noted that not all metrics are equally important and resulting decisions should normally not be based on a single metric but on a set of related metrics. Further, performance-related metrics need to be assessed in relation to usage parameters set by the Runtime Service Usage Thresholds precept (explained in Chapter 11).

Compliance Metrics

This type of metric can be collected manually or automatically. Manual compliance metrics are generally derived from the results of governance review processes (such as those described in Chapters 7–11). These results help determine the amount of compliance failures (or violations) that occur, which can then trigger corresponding vitality activities.

Automated compliance metrics are usually focused on runtime exceptions or logged events that occur when messages or data exchanged by services fail validation rules within technical policies, schemas, or other operational definitions. Alternatively, compliance metrics can apply when the usage of a service violates the parameters or limits set by the Runtime Service Usage Thresholds precept (see Chapter 11).

Organizational Shifts

This type of trigger occurs when internal changes are made to an organizational structure, or when other forms of internal events affect the IT enterprise, deliberately or inadvertently.

Examples of organizational shifts include:

- two departments are combined due to budgetary constraints
- a line of business is temporarily suspended due to a strike or labor protest
- there is a change in IT management, resulting in the reallocation of resources
- adjustments are made to the existing funding model

The Organizational Shifts trigger is differentiated from Strategic Adjustment triggers in that the organizational shift may not be the result of a change in strategic direction or policy. However, some strategic adjustments can lead to organizational shifts, in which case separately defined triggers may not be necessary.

Periodic

Periodic triggers are pre-scheduled, preventative reviews utilized primarily for quality assurance purposes. They are employed as a means of measuring and validating that an SOA ecosystem is performing as expected and in support of the overarching strategic direction. The following sections describe two common types of periodic triggers.

Milestone

A milestone trigger is associated with pre-defined vitality activities that are carried out at scheduled intervals. As an example, a series of reviews of a service inventory technology architecture may be scheduled to correspond with service quantity milestones. The first review is carried out upon the delivery of 10 services, the second review when the quantity reaches 20, and so on.

Time

Time triggers force a review of an asset using a pre-defined timescale. This facilitates the flexibility to define different classes of SOA assets with different time trigger durations. For example, every shared service deployed in a public cloud may be subjected to a statistics report on a monthly basis to collect usage and billing data. This information can then be used to compile ROI reports as the basis of a business metric.

SUMMARY OF KEY POINTS

- SOA governance vitality triggers represent common events and changes that force the execution of an SOA governance vitality process.
- Some vitality triggers are associated with business change, while others originate with technology change.
- Metrics are the most common form of vitality trigger.

13.3 SOA Governance Vitality Process

An SOA governance vitality process is comprised of a series of activities that are carried out in response to certain vitality concerns or requirements. Each of the previously described vitality triggers can result in different vitality activities (or different combinations of activities) being performed.

Figure 13.4 displays a generic sample process with five common types of vitality activities. The upcoming sections describe these activities in more detail. Note that vitality processes and activities need to be customized to ensure they address the specific needs of the organization.

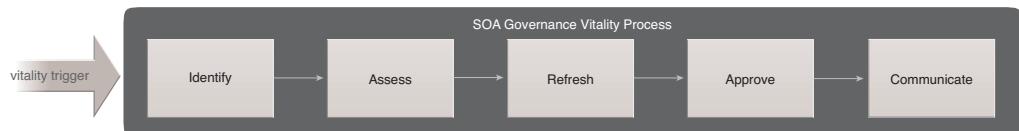


Figure 13.4

Five common activities as part of a generic SOA governance vitality process. This process is usually carried out manually by SOA Governance Specialists and other members of the SOA Governance Program Office.

Identify Activity

This base activity centers on capturing relevant information surrounding the trigger and the circumstances of its execution. The collected data can help determine which subsequent activities should be carried out and whether minor or more substantial vitality measures are required. Therefore, it is important that the identification criteria used by this activity is accurate and balanced.

Assess Activity

To enable the SOA Governance Program Office to make informed vitality decisions, it will need to assess the validity, impact, and value of each proposed refresh of an IT asset. The Assess activity encompasses tasks that focus on determining the effect of carrying out a refresh, prior to moving on to the actual Refresh activity.

Example assessment criteria includes:

- the motivation for the refresh
- concrete benefits of the proposed refresh
- degree of impact on existing and future projects
- the effect on other SOA assets
- the effect on consumers of the asset to be refreshed
- implications if the refresh is rejected
- legitimate alternatives to the refresh
- refresh limitations imposed by current deployment environments (such as when the environment is controlled by a third-party cloud provider)

Armed with this type of assessment information, those carrying out the SOA governance vitality process can make informed recommendations or decisions for approving, delaying, or rejecting proposed refresh requests.

Refresh Activity

The Refresh activity governs the actual actions performed in response to the previously identified vitality concern. As a task that is part of a governance process, this activity provides direction and regulation with regards to the necessary changes that need to be applied to the affected asset (or to the affected part of the SOA ecosystem).

Here are some examples:

- An SOA Governance Specialist carrying out the vitality process may develop a custom refresh plan that identifies the team, schedule, and budget required to perform necessary infrastructure upgrades to counter an identified scalability threshold.

- The Enterprise Architect involved with a vitality process identifies a flaw in an existing design standard and creates a plan to regulate the revision of the corresponding design standards specification.
- As a result of usage demands that exceed on-premise infrastructure capacity, the Service Custodian consults a Cloud Governance Specialist and they collectively decide to move the service implementation into a public cloud.
- Alternatively, a cloud-based service may have been the victim of repeated attacks that have compromised its performance. After receiving a report from the Cloud Resource Administrator that there is no immediate solution in sight, the Service Custodian decides to bring the service back to an on-premise environment.

Regardless of the nature of the required refresh tasks, the primary deliverable of this activity is generally a plan with proposed details for carrying out the actual refresh.

Approve Activity

The plan produced by the Refresh activity is subjected to an authority within the SOA Governance Program Office for formal approval. This activity is basically a review that can result in the acceptance or rejection of the refresh plan. Often, a rejection may simply be a request for more details or refinements.

In addition to cost and effort, a primary factor that weighs into this review process is the impact of the refresh on the existing SOA ecosystem. Because assets commonly affected by vitality activities are currently in production usage, impacts upon resources and consumers relying on the targeted assets can be significant. So much so, that there may be cases where legitimate refresh plans are rejected because they are not considered worth the disruption they would end up causing.

Communicate Activity

Once approved, the planned refresh needs to be communicated to the affected stakeholders, project teams, and others involved with the targeted asset (such as the Service Custodians).

Documents and reports need to be provided to ensure that all affected understand the objectives and consequences of the upcoming refresh. Note that this is communication that occurs *after* the Approve activity. The intent is to notify others of the refresh activities that will occur.

SUMMARY OF KEY POINTS

- An SOA governance vitality process is initiated as a result of the execution of a vitality trigger.
 - Each vitality process can be customized in relation to the nature of the trigger and the organization's requirements.
 - Common vitality process steps include Identify, Assess, Refresh, Approve, and Communicate.
-

Chapter 14



SOA Governance Technology

- 14.1** Understanding SOA Governance Technology
- 14.2** Common SOA Governance Technology Products
- 14.3** Guidelines for Acquiring SOA Governance Technology

It's no secret that SOA underwent an identity crisis during its earlier stages. As the acronym became an increasingly common part of IT media vocabulary, there were many associations and definitions. Most were wrong. One of the primary contributing factors to the resulting confusion was that some vendors wanting to capitalize on the hype began branding products with "SOA." Many trusting practitioners who invested in these products eventually failed in their attempts to realize project goals. Either the products were not legitimately supportive of SOA, or practitioners were led to believe that the purchase of the products alone would lead to the creation of a service-oriented enterprise, or both.

In the later stages of the SOA hype cycle, governance was being highlighted as the most common critical success factor overlooked by early adopters. This oversight was blamed as the reason for past failed projects and as the justification for organizations to "try to do it right" this time around. Unfortunately, the subsequent emergence of SOA governance technologies resulted in a related, yet separate stage of "mis-branding." This led to a new, yet familiar round of ambiguity and confusion.

In this chapter we establish what is and is not SOA governance technology. We describe common products used to support SOA governance systems for both design-time and runtime stages with reference to on-premise and cloud-based services. The chapter concludes with guidelines for choosing SOA governance products and vendors.

NOTE

While the upcoming products are discussed in relation to SOA governance, they are not necessarily exclusive to that purpose. For example, many of these products are used for hands-on management activities.

14.1 Understanding SOA Governance Technology

To fully understand what constitutes an SOA governance product and to further comprehend its meaning and purpose, we need to go back to the foundational definition of a governance system introduced in Chapter 6. A governance system is a meta-decision

system used by an organization to make decisions about decision-making. A governance system therefore places constraints on decisions, determines who has responsibility and authority to make decisions, establishes constraints and parameters that control, guide, or influence decisions, and prescribes consequences for non-compliance.

The purpose of the many governance precepts, processes, and roles throughout this book is to help you establish and maintain a governance system specific to SOA and service-orientation. It is this context we must remain aware of when exploring and assessing suitable technologies.

NOTE

In addition to the term “technology,” the terms “products” and “tools” are used throughout this chapter. Neither term is formally defined. A tool is most commonly a type of product with a front-end user interface. For the purpose of this chapter, both products and tools are considered forms of technologies.

SOA Governance Task Types

Let’s first begin with some common terms used to label types of SOA governance tasks. These categories will help us classify technologies based on their function and purpose.

Manual Governance

These are primarily decision-based tasks that need to be performed by people or as part of processes that are not fully automated. An example of a manual governance task is the review of a service architecture specification by an SOA architect.

Automated Governance

Automated governance tasks can include the collection of metrics used to support decision-making or tasks performed automatically to support or enforce governance precepts and processes. An example of an automated governance task is the use of a security mechanism to prevent access to a service architecture specification by an outside project team.

Design-time Governance

The “design-time” part of this term refers to manual or automated tasks performed prior to runtime service usage. An example of a design-time governance task is the review of modeled service candidates to ensure compliance to naming conventions. (SOA Adoption Planning and Service-Oriented Analysis stages are, for the purpose of this chapter, still considered design-time governance tasks.)

Runtime Governance

Runtime governance begins where design-time governance ends. This type of task therefore becomes primarily relevant after the service has been implemented for active usage. An example of a runtime governance task is the monitoring of a service’s runtime behavior to ensure compliance to published SLA guarantees.

NOTE

Runtime governance is often associated with the Service Usage and Monitoring stage. However, during the Service Deployment and Maintenance stage a service implementation moves from its pre-production to its production environment, making this stage relevant to both design-time and runtime governance controls.

On-Premise Governance

When the focus of governance activity is limited to controlled, on-premise environments, then it qualifies as on-premise governance tasks. An example of an on-premise governance task is the impact analysis performed by an SOA planning tool in order to determine the effect a new service version may have on existing service consumers and related artifacts within the on-premise service inventory.

Cloud Governance

Cloud governance tasks are those specific to governing services and related IT resources that are deployed in cloud environments. An example of a cloud governance task is the monitoring of per usage fees incurred by a SaaS deployment within a public cloud.

Passive Governance

This classification is used to label tasks that do not directly impact or interfere with the current state of a service or service-oriented solution. An example of a passive governance task is the collection and reporting of runtime or design-time metrics.

Active Governance

When governance tasks do have an immediate effect on the current state of a service or service-oriented solution, they are classified as “active.” An example of an active governance task is a review process, such as the compliance checking of a proposed XML schema. Upon failing compliance, the schema is rejected.

NOTE

A given governance task will belong to one of each of the previously described type pairs. For example, a task can be categorized as automated or manual, design-time or runtime, on-premise or cloud-based, and passive or active.

SOA Governance Technology Types

Let’s now establish some common categories that help group governance products based on their overall purpose.

Administrative

This category mostly represents tools used for manual governance purposes at design-time and runtime. An administrative tool is typically utilized to document and share published content pertaining to precepts and processes, or details about the governance system and the SOA Governance Program Office in general. Some forms of automated governance products may incorporate the usage of administrative functionality or may include a front-end tool allowing for administrative tasks.

Monitoring

A key means of measuring the effectiveness and success of an SOA governance system is to keep track of the performance of the people and solutions being governed. Monitoring products can range from front-end tools used to manually trace a service through its lifecycle stages to automated runtime technologies that observe services and service-oriented solution behavior. Monitoring products can carry out passive or active tasks, depending on how they are designed to respond to certain events or conditions encountered while monitoring.

Reporting

For governance professionals to assess and stay in touch with the design-time state of service delivery projects and the runtime performance of individual services, it is crucial to have reporting mechanisms in place that collect and communicate various metrics and statistics. This information is commonly used by the SOA Governance Program Office as vitality input for evolving the governance system.

Enforcement

Those participating in manual processes are often assigned the responsibility of checking for compliance to SOA governance precepts. Depending on the nature of the compliance-checking being performed, this type of active governance task can be supported by front-end tools and back-end technologies that help validate or automatically determine compliance or non-compliance.

NOTE

There can be a natural overlap between products that provide monitoring and reporting features. Depending on the nature of the information being gathered, reporting functionality often needs to encompass an extent of monitoring.

NOTE

The Service Discovery stage can also be considered a design-time stage when an existing agnostic service (that has likely already been active in runtime) is discovered during the design-time stages of a new project.

SUMMARY OF KEY POINTS

- There are eight task types that can be used to classify SOA governance products based on how and when they are typically applied in support of governance activities.
 - There are five technology types that can be used to categorize SOA governance products based on the nature of their functionality.
-

14.2 Common SOA Governance Technology Products

The upcoming section contains descriptions for a set of primary products, each of which further includes sub-sections that map a product to related task types and technology types, as well as commonly associated SOA project stages. How and where technologies can be involved can vary greatly, depending on the nature of the governance system and the scope and reach of the SOA Governance Program Office.

Specifically, the following SOA governance technologies and products are covered:

- Service Registries
- Repositories
- Service Agents
- Policy Systems
- Quality Assurance Tools
- SOA Management Suites

It's important to acknowledge that other types of governance-branded products exist. The listed products were chosen because they are established and have proven themselves valuable specifically in support of SOA governance systems.

NOTE

For many examples of current vendor products for each of the upcoming categories, visit www.soabooks.com/governance/. The screenshots and information published at this part of the book series Web site is intended to help readers associate the described SOA governance technology product categories with real world products. Vendors have an open invitation to freely contribute content to this part of the Web site; however, none of the products displayed are endorsed by the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*.

Service Registries

A service registry is used to store metadata about services, including descriptions of their functional contexts and capabilities, as well as pointers to the locations of their service contracts.

Private and public service registries can exist, depending on the scope of intended consumers. A private service registry is typically intended for access by consumers within the IT enterprise and is often specific to a particular service inventory.

To locate services, people typically search service registries using specialized query tools. A number of different technologies have been employed to provide service registry functionality, from relational databases to flat files registry to products compliant with the Universal Description, Discovery, and Integration (UDDI) industry standard.

The usage of a service registry is mandated by SOA governance precepts and related processes, most commonly in relation to discovery requirements (as explained in Chapter 10). The metadata within the service registry can further support the SOA Governance Program Office by providing a central data source used to report on the status of upcoming and existing services, as well as the overall status of an entire service inventory.

Task Types

The service registry can be used for all forms of SOA governance tasks. Members of the SOA Governance Program Office can contribute to or maintain registered service metadata the same way project teams delivering the service can. Metadata can be added or updated at design-time and further updated and accessed at runtime. The data can be used for passive reporting purposes, or it may lead to the need for responsive, active governance action. Although service registries can exist in cloud platforms, it is more common for them to be positioned in on-premise environments, with pointers to cloud-based services.

Manual	x
Automated	x
Design-time	x
Runtime	x
On-Premise	x
Cloud	x
Passive	x
Active	x

Technology Types

The recording and maintenance of service metadata within the service registry can be classified as an administrative task. Some service registries are equipped with built-in reporting features (comparable to search results), and the structure of the service records within the registry can force compliance to service registry standards and conventions. Furthermore, the usage of the service registry itself can be (and often is) the focal point of a primary compliance requirement for project teams.

Administrative	x
Monitoring	
Reporting	x
Enforcement	x

SOA Project Stages

Because service metadata can be added and updated throughout the project and service lifecycles, the service registry can be associated with all lifecycle stages. However, it is most commonly known for its association with the Service Discovery stage.

Besides acting as a central administrative resource for users to keep track of the status of services, a primary function it fulfills is to provide access to service metadata for project teams that need to locate (discover) agnostic services for reuse purposes.

SOA Adoption Planning	x
Service Inventory Analysis	x
Service-Oriented Analysis	x
Service-Oriented Design	x
Service Logic Design	x
Service Development	x
Service Testing	x
Service Deployment & Maintenance	x
Service Usage & Monitoring	x
Service Discovery	x
Service Versioning & Retirement	x

Repositories

An SOA repository is a specialized database used as a storage mechanism for a range of design-time artifacts that can pertain to individual services or entire service-oriented solutions. Examples of artifacts commonly stored in repositories include:

- service profiles and design specifications
- programming code, including executable business process logic (such as WS-BPEL)
- data models (such as those defined using the XML Schema Definition language)
- technical service contract documents (such as WSDL and policy definitions)
- human-readable service contract documents (such as SLAs)
- business analysis documents (such as process models)

In addition to serving as a general storage container, repository products are also often capable of automatically parsing artifacts to determine relationships, as well as enabling additional input. Users interact with repositories the same way they do with service registries—by using query tools and user-interfaces provided by the repository vendor.

Although most of the artifacts stored in a repository are technical in nature and administered by Service Architects and Service Developers, Service Custodians and SOA Governance Specialists also can contribute to and maintain certain types of artifacts, most notably service profiles and SLAs.

Task Types

The typical artifacts stored in a repository are most relevant to both manual and automated governance tasks that occur at design-time. This is because the repository simply provides central storage for related documents and technical artifacts that are most used during and relevant to the lifecycle stages that lead up to service implementation and runtime usage. Although it is technically possible for repositories to be used to store these types of artifacts in cloud-based environments, they are more commonly kept on-premise where they can be centrally administered. (An exception may be the use of a private cloud to make a repository accessible to other internal cloud consumers. In this case, a Cloud Storage Specialist may become involved to determine the best repository implementation and access methods.)

Manual	x
Automated	x
Design-time	x
Runtime	
On-Premise	x
Cloud	
Passive	x
Active	

Technology Types

The storage and maintenance of documents and artifacts within the repository are considered administrative governance tasks, especially as they serve the purpose of making the repository contents centrally shareable by a range of project team members. Some repository products are capable of reporting various types of information about stored artifacts in relation to services and service composition, and more recent repositories can even parse and validate some forms of artifacts (such as service contract definitions). Furthermore, the repository can be subject to access control limitations to help SOA Governance Specialists prevent unauthorized project teams from gaining access to confidential service implementation information.

Administrative	x
Monitoring	
Reporting	x
Enforcement	x

SOA Project Stages

Just about any form of documentation and technical component can be stored in a repository. Therefore, this type of SOA governance technology can be potentially associated with all service lifecycle stages.

The Service Discovery stage is also included as it is common for project teams, subsequent to identifying agnostic services they would like to reuse as part of a new solution, to request further access to technical and human-readable artifacts. For example, after choosing a service listed in a service registry, an SOA architect on a project team may then want to retrieve the service's actual service contract definitions.

NOTE

Several vendors provide products that combine service registry and repository features.

SOA Adoption Planning	x
Service Inventory Analysis	x
Service-Oriented Analysis	x
Service-Oriented Design	x
Service Logic Design	x
Service Development	x
Service Testing	x
Service Deployment & Maintenance	x
Service Usage & Monitoring	x
Service Discovery	x
Service Versioning & Retirement	x

Service Agents

A service agent is an event-driven software program capable of passively tracking runtime activity and actively responding when encountering pre-defined conditions. Service agents are a common part of SOA governance products that contain back-end technology and processing logic.

Typical functions carried out by service agents include:

- validation
- enforcement
- monitoring
- notification
- logging

- exception handling
- billing and payment data collection
- dynamic scaling

A service agent is different from a service in that it does not provide a published contract and is therefore not explicitly invoked. Further, the type of logic encapsulated by service agents is generally utility-centric.

Service agents can be custom-developed to support various types of runtime governance requirements. A popular example is a service agent that issues a notification each time it detects a violation of a runtime governance precept. These notifications can be logged, forming the basis of a regular report published for the SOA Governance Program Office.

NOTE

For an introductory description of service agents, see the definition in Chapter 3.

Task Types

Service agents are specifically used for automated governance tasks that occur at runtime. Service agent logic can be designed to carry out both passive and active governance tasks, depending on how the service agent is utilized and also on whether the service agent is part of a larger product environment or was custom-developed for specific governance automation requirements. Note that some service agents can perform a range of processing, whereby only a subset of this functionality may actually be directly related to supporting SOA governance activities.

Cloud environments, in particular, rely heavily on the use of service agents that are native parts of cloud platforms. For example, cloud-based service agents are commonly used to support administrative cloud services that expose APIs or provide user interfaces for the configuration and reporting of monitoring and diagnostic functions. Some of these agents can dynamically seek and collect instrumentation information about active cloud services or entire cloud-based service compositions.

Manual	
Automated	x
Design-time	
Runtime	x
On-Premise	x
Cloud	x
Passive	x
Active	x

Technology Types

As previously discussed, common types of utility-centric processing provided by service agents include monitoring, reporting, and enforcement. Cloud-based service agents further focus on billing and on-demand scaling capabilities. These forms of runtime functions can be used to assist with a wide range of SOA governance tasks.

Administrative	
Monitoring	x
Reporting	x
Enforcement	x

SOA Project Stages

Although development tools and various design-time platforms may employ service agents in support of common SOA governance tasks, they are primarily utilized for runtime processing and are therefore used to passively observe and/or actively respond to runtime conditions.

For example, service agents may keep track of requests or inquiries issued against a service registry in order to collect discovery-related metrics. Similarly, service agents may be used to monitor the usage of a particular service version, especially when a new version of the service co-exists with the old version.

SOA Adoption Planning	
Service Inventory Analysis	
Service-Oriented Analysis	
Service-Oriented Design	
Service Logic Design	
Service Development	
Service Testing	
Service Deployment & Maintenance	
Service Usage & Monitoring	x
Service Discovery	x
Service Versioning & Retirement	x

Policy Systems

A typical policy system provides three types of functions: policy definition, policy enforcement, and policy monitoring. This type of governance product generally provides front-end UIs that enable you to create policy logic that is then exported in the form of technical policy definitions. The back-end of the system then makes use of

service agents to monitor policy processing activity and, when necessary, enforce the policy conditions.

When policy violations occur, the service agents can be configured to actively or passively respond by either preventing the activity from completing or merely issuing a notification that the violation happened. Policy systems allow for the central maintenance of collections of policies, often for an entire service inventory. Some systems even tie in to service registries and repositories.

NOTE

For introductory coverage of operational and business policies, see the *Policies 101* section in Chapter 12.

Task Types

As technical artifacts that exist as part of back-end architectures, policies can be considered a runtime technology comparable in scope to service agents. Because a policy system will typically employ the use of specialized service agents, policies (and the service agent logic used to validate and enforce them) can result in passive or active responses to various runtime conditions.

Manual	
Automated	x
Design-time	
Runtime	x
On-Premise	x
Cloud	x
Passive	x
Active	x

Technology Types

As runtime artifacts, the usage of policies primarily pertains to runtime processing, which can encompass monitoring and enforcement. Further, various policy metrics (especially compliance-related) form the basis for reporting input.

Administrative	
Monitoring	x
Reporting	x
Enforcement	x

SOA Project Stages

A policy system is comprised of policy definitions that can be business or utility-centric in the type of logic they contain. The logic behind policies can be collected as early as the analysis stages when service candidates are derived from various forms of business intelligence. Although policy definitions will typically be positioned as extensions of service contracts, the results of compliance checks can impact how the underlying service processing logic needs to respond.

SOA Adoption Planning	
Service Inventory Analysis	x
Service-Oriented Analysis	x
Service-Oriented Design	x
Service Logic Design	x
Service Development	x
Service Testing	x
Service Deployment & Maintenance	x
Service Usage & Monitoring	x
Service Discovery	x
Service Versioning & Retirement	x

Quality Assurance Tools

When services pass through the SOA project stages they can be subjected to multiple governance review processes, each of which is responsible for ensuring compliance to certain precepts. Many of these compliance checks are related to verifying a baseline measure of quality as it pertains to the service's compatibility with its platform and surrounding services, as well as the service itself.

Prior to being implemented in the production environment, the service must undergo testing and other forms of quality assurance controls as part of the Service Testing life-cycle stage. Tools are typically used by SOA Quality Assurance Specialists to perform various tests and to collect information for subsequent test reports.

Of the sample precepts associated with the Service Testing stage (listed in Chapter 10), the Testing Parameters Standards and Service Model Testing Standards represent

a set of regulations that can be incorporated within customizable quality assurance tools. This can make steps of the Service Test Review process automated or, even if still manual, more seamlessly part of processes carried out by SOA Quality Assurance Specialists.

It can be further valuable to the SOA Governance Program Office for metrics to be collected at this stage to report on how well services responded to various types of tests. For example, if a high level of failure for a certain type of test is encountered, SOA Governance Specialists can consider whether new precepts should be introduced to help alleviate these types of recurring problems.

Task Types

Quality assurance tools are generally used at design-time during which, for governance purposes, both manual and automated tasks can be carried out. The focus of the processing performed by these tools is on compliance checking and therefore, the results of failed or successful checks can be configured. Some forms of checking will demand an active response that results in the service (or part of the service architecture) being rejected. Passive responses are also possible, especially when compliance is optional. Cloud-based quality assurance tools are commonly limited to reporting functions with primary governance tasks being carried out on-premise (especially when the cloud environment itself is the focal point of quality assurance concerns).

Manual	x
Automated	x
Design-time	x
Runtime	
On-Premise	x
Cloud	x
Passive	x
Active	x

Technology Types

Depending on how a quality assurance tool is customized to support SOA governance precepts and processes, it may assume various types of reporting functions to provide metrics and other forms of reports of particular interest to the SOA Program Governance Office. Enforcement is the natural task performed by these tools; however, not all forms of service testing enforcement are necessarily pertinent to SOA governance precepts.

Administrative	
Monitoring	
Reporting	x
Enforcement	x

SOA Project Stages

Due to the nature of the processing performed by quality assurance tools, their involvement is generally limited to the Service Testing stage.

SOA Adoption Planning	
Service Inventory Analysis	
Service-Oriented Analysis	
Service-Oriented Design	
Service Logic Design	
Service Development	
Service Testing	x
Service Deployment & Maintenance	
Service Usage & Monitoring	
Service Discovery	
Service Versioning & Retirement	

SOA Management Suites

Several SOA product vendors have made large-scale SOA management systems available, some of which can handle environments containing thousands of services, comprising hundreds of compositions that exchange vast amounts of messages per second.

Such systems can provide broad feature sets, including:

- governance impact planning
- service administration
- service monitoring
- service mediation
- auditing and logging
- service management
- error diagnosis and remediation

An SOA management system is often comprised of a set of products and therefore also referred to as a management suite. The products in the suite tend to be more about the hands-on management of services and their respective runtime environments, meaning that if we revisit the distinction we made back in Chapter 6 between management and governance, these systems are primarily relevant to the former. However, because of the all-encompassing nature of these solutions, they often contain features suitable for SOA governance purposes.

NOTE

Because SOA management suites can vary significantly in terms of feature-sets and relevancy to SOA governance support, the *Task Types*, *Technology Types*, and *SOA Project Stages* sections are not provided. If they were shown, all items in all three tables would be checked off.

Other Tools and Products

There are no limitations as to the types of technologies that can be used for SOA governance purposes. As long as their usage legitimately supports SOA governance precepts and processes, they can be considered SOA governance technologies.

Provided here are further common examples.

Technical Editors and Graphic Tools

Several of the review processes and compliance-related precepts introduced by an SOA governance program can require validation of technical data. In order for SOA Governance Specialists to perform this validation, there may be the need to use tools capable of displaying the data in a structured or graphical manner. Some technical editors have the ability to render programming code into a graphical representation, which can make a manual audit easier to perform.

Content Sharing and Publishing Tools

Communication of SOA governance regulations and related information is extremely important to the success of an SOA Governance Program Office. Within the office, new or updated precepts and processes need to be centrally maintained and published for access to all that work within or in relation to the SOA Governance Program Office. Furthermore, having an open, two-way communications channel between the SOA

Governance Program Office and other departments and project teams within the organization is vital for SOA Governance Specialists to measure and assess the effectiveness of the SOA governance system.

Traditional Web-based content sharing mediums, such as intranets, content management systems, or document sharing and versioning systems, can be used to ensure consistent and accurate dissemination of governance information. Ideally, IT professionals outside of the SOA Governance Program Office are given the ability to subscribe to the SOA governance program so that they are automatically notified of changes or additions to the SOA governance system.

Configuration Management Tools

The same way that quality assurance tools can be used to support SOA governance precepts and processes specific to the Service Testing lifecycle stage, configuration management and versioning tools can be utilized in relation to precepts and processes established to help regulate the Service Versioning stage. Note that these tools can provide metrics for versioning issues pertaining to technical service contracts as well as SLAs. One of the metrics of most interest to the SOA Governance Program Office is the frequency of service or service contract versions, as this can shed light on flaws or problems that are inhibiting the longevity of services.

Custom SOA Governance Solutions

When vendor or open-source products are insufficient or, for other reasons, not the correct choice for adding SOA governance technology to your environment, there always exists the option of building your own. Sometimes this approach is justified, especially when your organization's governance requirements are so distinct that commercially available technologies are simply not sufficient.

When the choice between existing products and custom solutions does exist, it is important to weigh the consequences of each before deciding on a direction. Although SOA governance products are by no means inexpensive, it is generally significantly more costly to develop and then maintain your own solutions, especially when having to scale those solutions in tandem with a growing inventory of services.

Furthermore, factors such as security, reliability, and expected ROI need to be addressed. In many cases, it makes more sense to find a packaged product that can be customized to an extent that most technical SOA governance requirements are fulfilled. But, if your organization is accustomed to building robust software programs (especially if your

line of business is already the delivery of packaged software), then you may be able to leverage existing resources and expertise to create and evolve the best possible governance technologies and tools for your requirements.

SUMMARY OF KEY POINTS

- Any technology can be considered an SOA governance technology if it can be used to effectively support the precepts and/or processes defined for a given SOA governance system.
 - Common SOA governance technologies include service registries, repositories, service agents, policy systems, quality assurance tools, and SOA management suites.
-

14.3 Guidelines for Acquiring SOA Governance Technology

Provided in this final section are some general strategies and best practices for choosing SOA governance products.

Acquisition Strategies

How you go about choosing your SOA technology vendor will have long-term consequences, not just in relation to how you will (and will not) be able to govern your service inventory, but also as to how cost-effective and effort-intensive that governance responsibility will be.

There are four common approaches that you can take: single vendor, multiple vendors, open source, and leasing from a cloud vendor. Let's take a closer look at each.

Single Vendor

Purchasing all (or the majority) of your SOA governance technology from a sole vendor can simplify the acquisition process, but can also impose some long-term limitations. The following lists explore the pros and cons.

Pros

- *Standardization* – By sticking to a single vendor, it will be easier to standardize SOA governance processes as well as the usage of the governance products themselves.

All SOA Governance Specialists working for the SOA Governance Program Office can be required to use the same line of products.

- *Skill-Set* – With a single source for SOA governance products, it will be easier to train and develop proficiency among the governance team, as well as others that may want to use the products for non-governance activities.
- *Seamless Integration* – If your technical SOA governance platform is comprised of various moving parts, having them all belong to the same product line will make any required integration significantly easier than if they came from different vendors.

Cons

- *Vendor Lock-In* – Vendor products are often designed to encourage customers to form dependencies upon the vendor. Entrusting just one vendor with all of your SOA governance software needs will likely lock you into the direction they choose for their product line. This may end up inhibiting your ability to evolve your SOA governance precepts and goals.
- *Vendor Credibility* – Some vendor organizations are acquisition targets for other, larger product vendors. If you have invested only in one vendor product line and the vendor company is acquired, it could have serious impacts on future support and product line direction.
- *Integration Issues* – As previously stated, a typical advantage to this approach is improved integration. However, that is not always the case. Some vendors provide SOA governance platforms that appear (from a packaging perspective) to be uniform, but are in fact assembled from a hodgepodge of disparate products, some of which may have been the result of corporate acquisitions.

Multiple Vendors

This is the best-of-breed approach, whereby you pick and choose the most suitable SOA governance products from different vendors. The result is that you intentionally create a heterogeneous SOA governance platform.

Pros

- *Leverage Innovation* – By deciding to assemble your SOA governance environment from the best possible solutions, you're much more likely to end up with highly sophisticated feature-sets capable of maximizing your requirements potential.

- *Independence* – By not committing to any single vendor vision and product road-map, you greatly increase your freedom to evolve your SOA governance platform in response to future needs.
- *Cost-Effectiveness* – This approach may more easily enable you to generate competition among vendors in order to drive down costs.

Cons

- *Integration Challenges* – The disparity within a diverse SOA governance platform will most likely increase the cost and effort required to make different governance tools and products connect. This further increases the likelihood that custom development may be required. These types of integration challenges can be further compounded when different vendors choose to take new directions with their products over time.
- *Skill-Set* – Supporting a range of products from different vendors can be burdensome when having to train staff to use and maintain multiple products.

Open Source

Some SOA governance technologies (and products providing governance-related functionality) exist as open source offerings. An approach can be used whereby the SOA governance platform is assembled entirely or partially from open source technology.

Pros

- *Usually it's Free* – The common basis of the open source model is that the software is made available to the community at no charge. This, of course, makes an open source SOA governance product more desirable from a budgetary perspective.
- *Access to Source Code* – Another common basis of the open source model is that the entire source code of the software is also made available to the community. The advantage here is that you are able to extend and fully customize the product as long as you have sufficient development resources.

Cons

- *Extra Support Agreements* – Another common revenue stream for open source organizations is support contracts. Because you will likely be relying on your SOA governance software to oversee vital daily operations, it's quite possible that you'll decide to pay for a support plan.

- *Lack of Features* – As we'll describe in a moment, open source solutions may offer noticeably less functionality than their proprietary brethren. In some cases, not having access to these important capabilities may outweigh the cost savings of an open source package.
- *Reliability Problems* – Because open source products may not be subjected to the same quality assurance rigor as commercial software products, they can be less stable. This can lead to serious consequences (or a significant maintenance burden) when relying on these products for SOA governance purposes.

Leased from Cloud Vendor

Various cloud providers offer virtualized desktop tools that can assist with cloud governance tasks. These are used for the remote governance and overall administration of cloud-hosted services and supporting IT resources.

Pros

- *You Don't Need to Buy* – Instead of having the up-front cost of purchasing governance technology, you can lease it as part of a licensing package with the cloud provider.
- *Optimized to Environment* – Generally, a cloud provider will offer tools that are very compatible to the nature of the cloud environment hosting the services and IT resources you would want to perform the governance tasks on.

Cons

- *Limited Options* – By working with the governance tool(s) a cloud provider makes available, you may be limited as to the functionality and extent of the governance activities you can actually perform.
- *Shared Hosting* – The governance tools you may be remotely accessing will commonly be virtualized and hosted by shared servers that can impact performance and reliability.

Of course, depending on the location of services within a given service inventory and the extent of additional on-premise governance tasks required, it may be necessary to combine leased cloud-based governance tools with on-premise governance tools.

Best Practices

The following best practices can form the basis of a checklist for assessing SOA governance vendors and products.

Establish Criteria Based on Your Specific Requirements

SOA governance technology is best evaluated after you have documented a comprehensive draft of your planned SOA governance system, along with the various parts of the supporting SOA governance program and even the organizational structure of the SOA Governance Program Office. With all of this in place, you will be able to define very specific criteria for technologies that can help realize the goals of the SOA governance program. It is with this criteria in hand that you will be in the best position to judge the applicability and usefulness of the various features offered by different SOA governance systems, products, and platforms.

Investigate Customizability

Some vendor products are extremely sophisticated, but allow for little opportunity to further change how they function or what boundaries they can function within. Regardless of whether you initially need to perform any major customization work on an SOA governance product, be sure to fully understand the extent to which the product can, in fact, be tailored. The more flexible the product is with regards to accommodating a range of requirements or parameters or even governance styles, the greater the chances that the investment you are making in the product today will last.

Investigate APIs

SOA governance product integration is often a key success factor to the long-term usage of an SOA governance platform. If products can be easily connected to share data and functionality, the platform can more responsively be evolved to accommodate new and changing governance requirements. If identified integration points result in actual integration and development projects, the SOA governance platform itself may be in need of its own system of governance.

Understand Both Initial and Long-Term Costs

The immediate costs required to establish SOA governance products, along with the long-term implications of licensing costs, need to be fully understood and compared before deciding on a given product. It is very helpful to have a service inventory blueprint at hand to fully assess the potential scope at which governance technology may

need to be deployed and utilized. Estimates pertaining to scalability of products (in terms of concurrent users and runtime capacity) together with supplementary expenses (such as training and integration) can form the basis of cost projections that help reveal required budgets.

Understand Actual Governance Support

As discussed earlier in this chapter, what constitutes a product that provides true SOA governance-related features can vary, depending on how vendors choose to brand and market their offerings. The best approach is, with your requirements criteria already defined, to disregard brands and labels and focus on the actual features and functions provided. Sometimes a subset of SOA governance product features will be relevant to your requirements and in other cases products not labeled with “SOA governance” at all will contain useful features.

Take the Time to Create a Quality RFP

Whether you are following the single vendor or multiple vendor strategy described earlier, it is worthwhile to put together a comprehensive RFP before approaching any vendor with your requirements. The most effective RFPs are created with the involvement of numerous IT professionals, not just the SOA Governance Specialists who may be leading the acquisition effort. It’s further important that the RFP be the culmination of an honest, unbiased assessment of your SOA governance requirements and expectations.

SUMMARY OF KEY POINTS

- When choosing SOA governance technologies, a single vendor, multiple vendor and/or open source acquisition approach can be considered—or—you may be required to lease governance tools from a cloud provider.
 - Key best practices for deciding on the most suitable SOA governance products focus on ensuring compatibility with your specific governance requirements and fully understanding the extent to which a product can be customized and integrated with other products.
 - A fundamental criterion for assessing SOA governance offerings is to have a clear understanding of what actually constitutes SOA governance activity within your organization.
-

This page intentionally left blank



Part IV

Appendices

Appendix A: Case Study Conclusion

Appendix B: Master Reference Diagrams for Organizational Roles

Appendix C: Service-Orientation Principles Reference

Appendix D: SOA Design Patterns Reference

Appendix E: The Annotated SOA Manifesto

Appendix F: Versioning Fundamentals for Web Services and REST Services

Appendix G: Mapping Service-Orientation to RUP

Appendix H: Additional Resources

This page intentionally left blank

Appendix A



Case Study Conclusion

When Raysmore was still in the planning stages of their SOA adoption project, they performed an Organizational Governance Maturity Assessment (see the case study example at the end of Chapter 7) whereby the following areas were evaluated:

- Cultural Readiness
- Centralization Factors
- Political Environment
- Technical Project Roles

The results of the initial assessment were not sufficient to proceed with the project at that time. Subsequent efforts to improve communication and education among affected departments and personnel led to a follow-up assessment with scores high enough to proceed.

While SOA Governance Specialists were in the midst of coordinating and documenting necessary vitality precepts (Chapter 11), they reflected on how the initial two maturity assessments differed after a concentrated effort to address identified shortcomings. If pre-project improvements in communications and skill-set development among project teams could increase organizational maturity, then what about the actual hands-on experience of carrying out the project with governance controls and the experience of the on-going governance of deployed services? This type of real-world exposure should, in theory, further help affect and help evolve the organization's maturity.

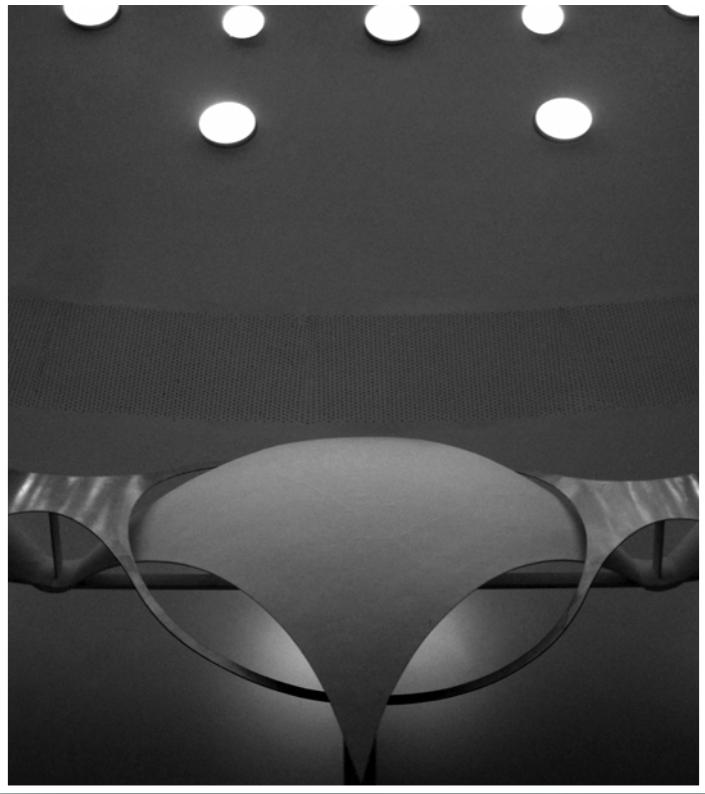
It is therefore decided to add a new scheduled vitality trigger. The same Organizational Governance Maturity Assessment will be carried out annually to determine the following:

- areas where organizational maturity has continued to grow
- areas where organizational maturity has begun to regress

Scores collected will help identify upward and downward trends in any of the measured areas. This, in turn, will help assess the effectiveness of precepts and processes that address or impact those areas.

Finally, this yearly assessment will highlight how the repeated application of SOA governance precepts and processes themselves will influence and shape different aspects of the organization, its culture, its technology, and its success in leveraging governed services in support of realizing business goals.

This page intentionally left blank



Appendix B

Master Reference Diagrams
for Organizational Roles

This appendix contains a series of reference diagrams that illustrate the mapping between organizational roles and SOA governance precepts and processes. Each identified precept and process is further labeled with its chapter of origin.

Service Analyst

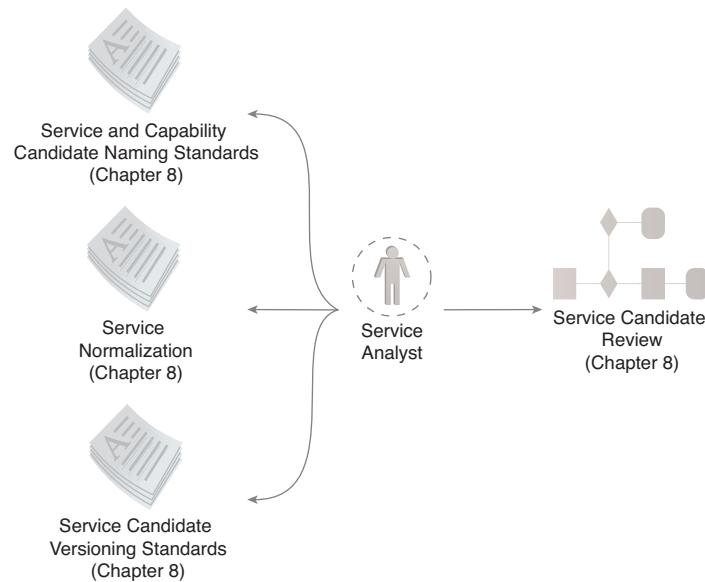


Figure B.1

SOA governance precepts and processes associated with the Service Analyst role.

Service Architect

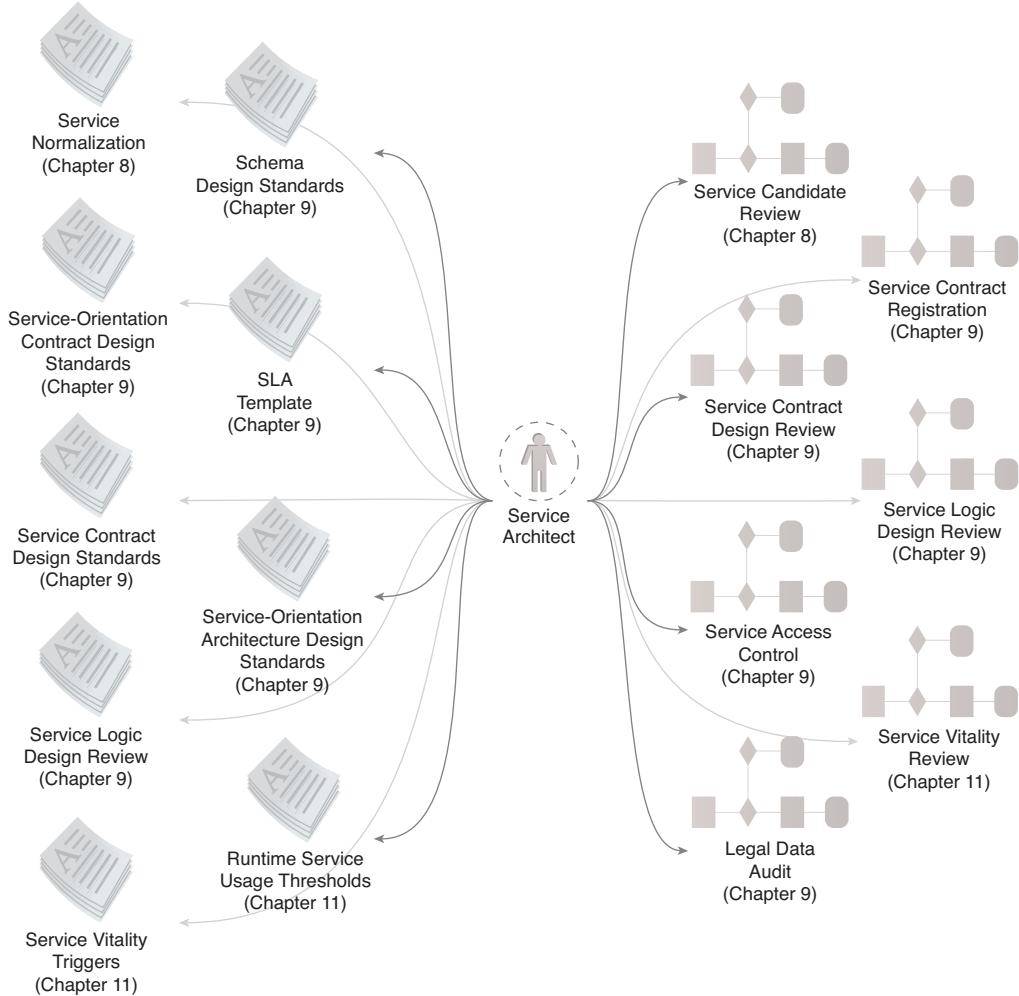


Figure B.2

SOA governance precepts and processes associated with the Service Architect role.

Service Developer

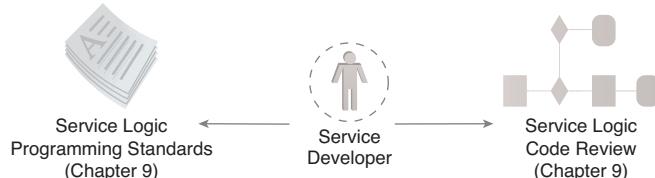


Figure B.3

SOA governance precepts and processes associated with the Service Developer role.

Service Custodian

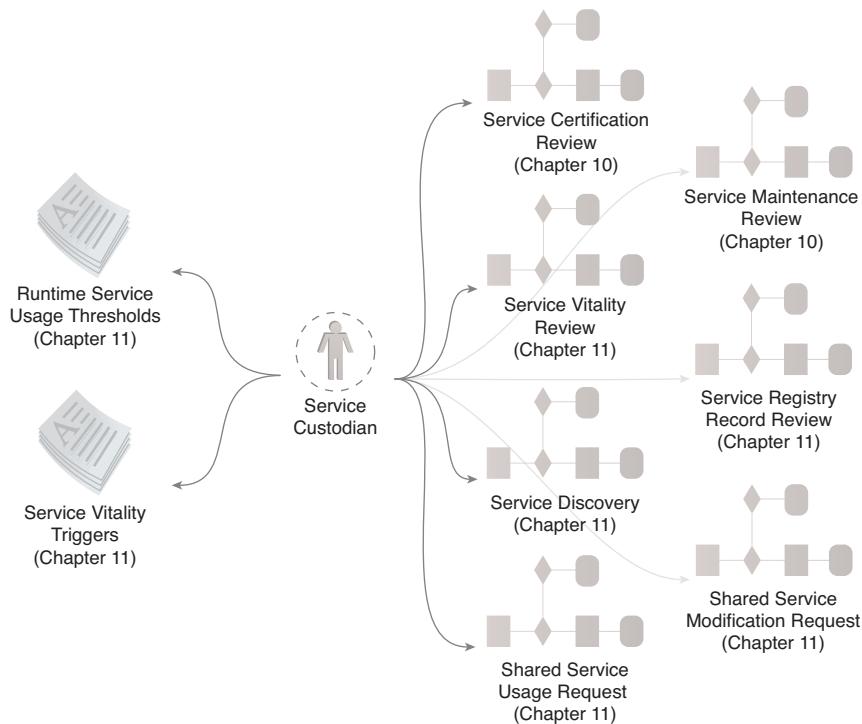


Figure B.4

SOA governance precepts and processes associated with the Service Custodian role.

Service Administrator

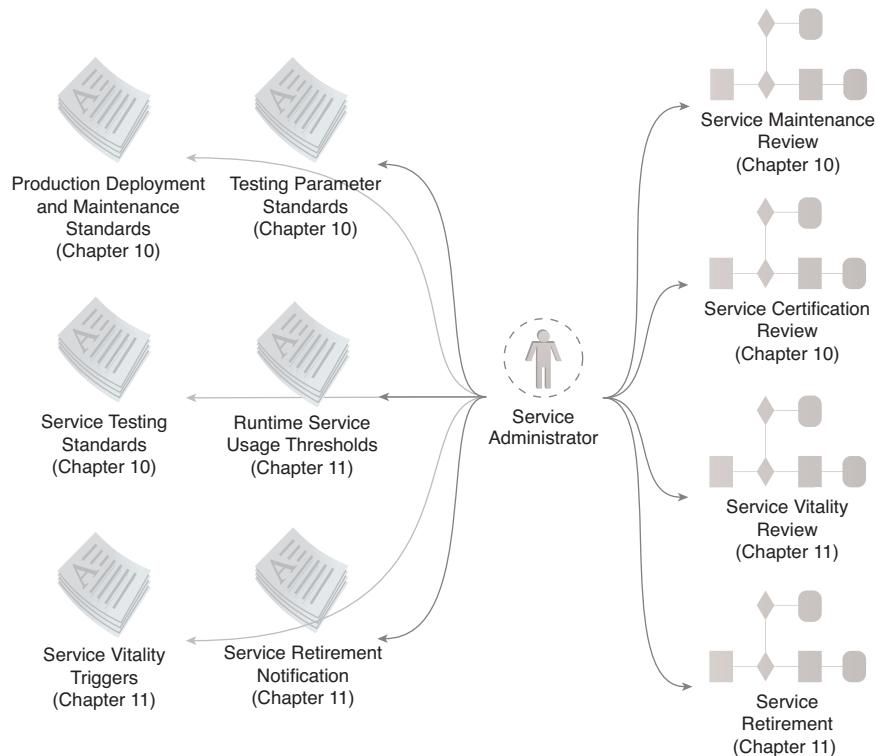


Figure B.5

SOA governance precepts and processes associated with the Service Administrator role.

Cloud Resource Administrator

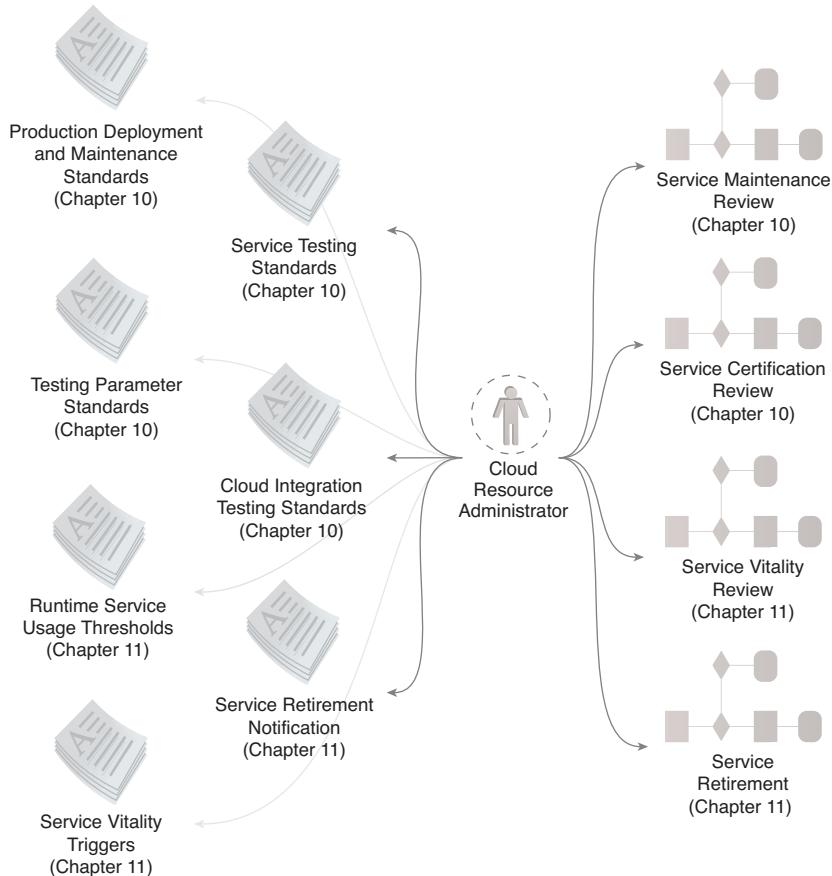


Figure B.6

SOA governance precepts and processes associated with the Cloud Resource Administrator role.

Schema Custodian

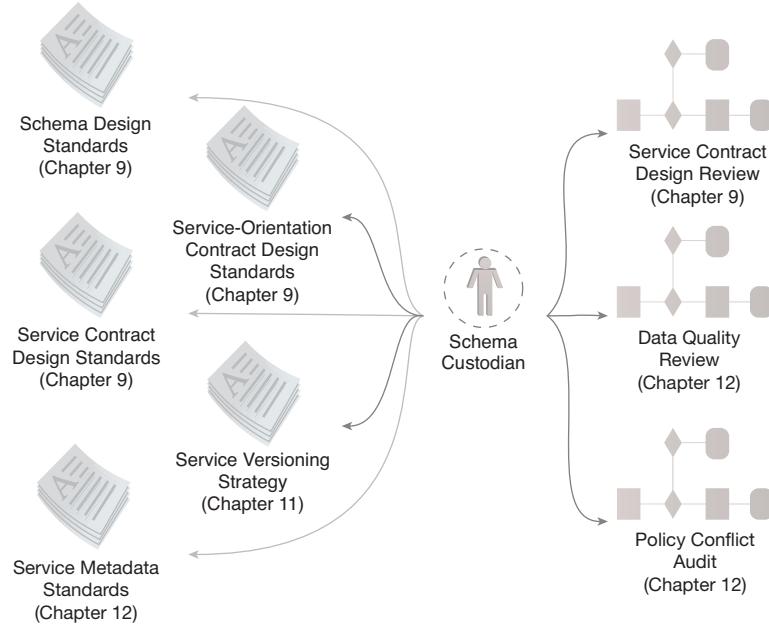


Figure B.7

SOA governance precepts and processes associated with the Schema Custodian role.

Policy Custodian

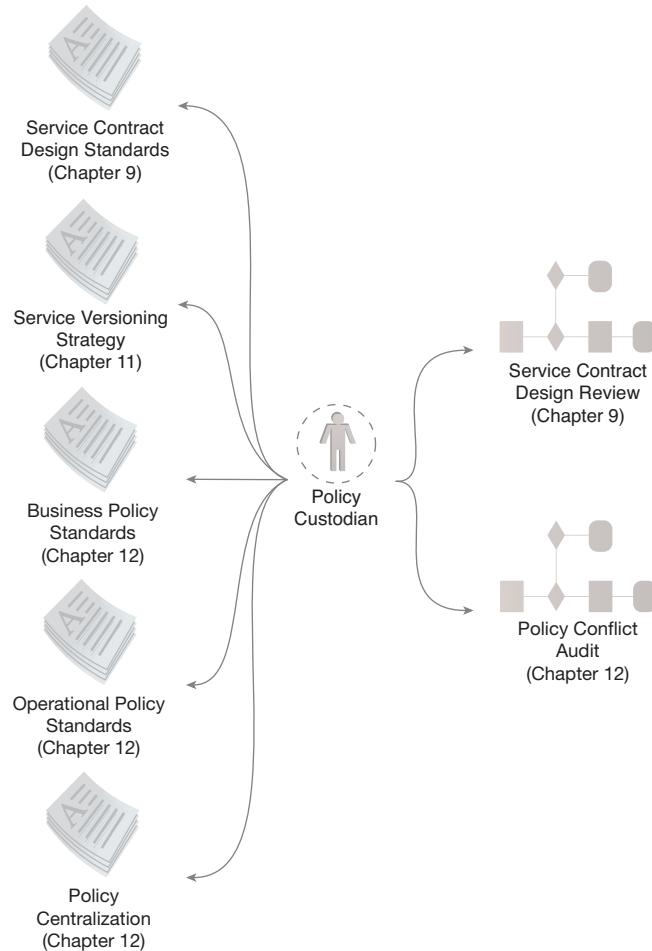


Figure B.8

SOA governance precepts and processes associated with the Policy Custodian role.

Service Registry Custodian

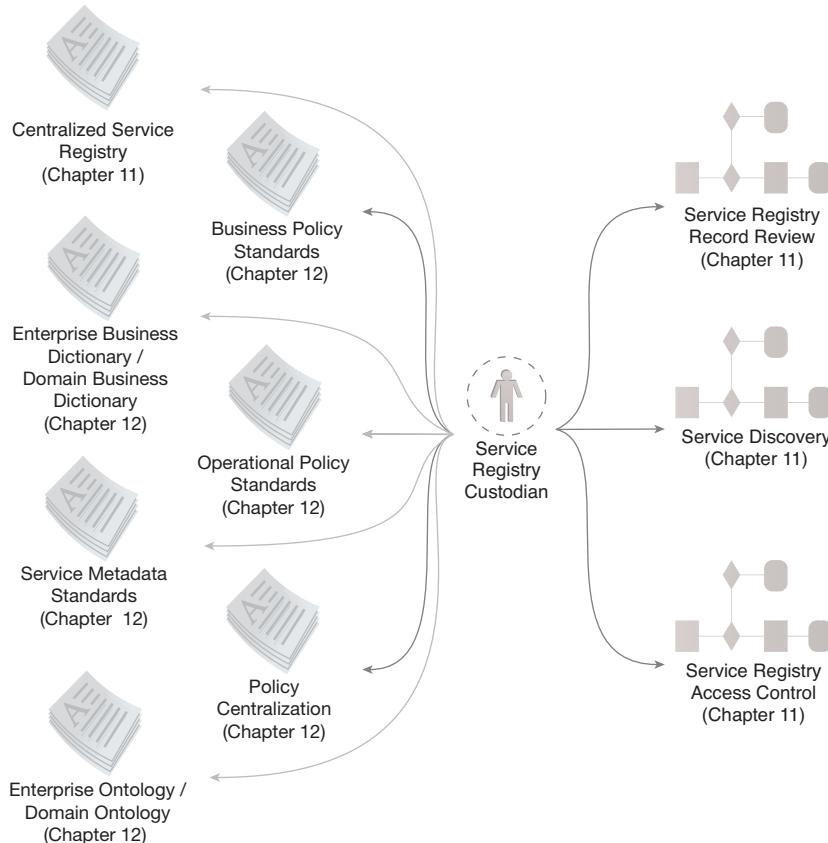


Figure B.9

SOA governance precepts and processes associated with the Service Registry Custodian role.

Technical Communications Specialist

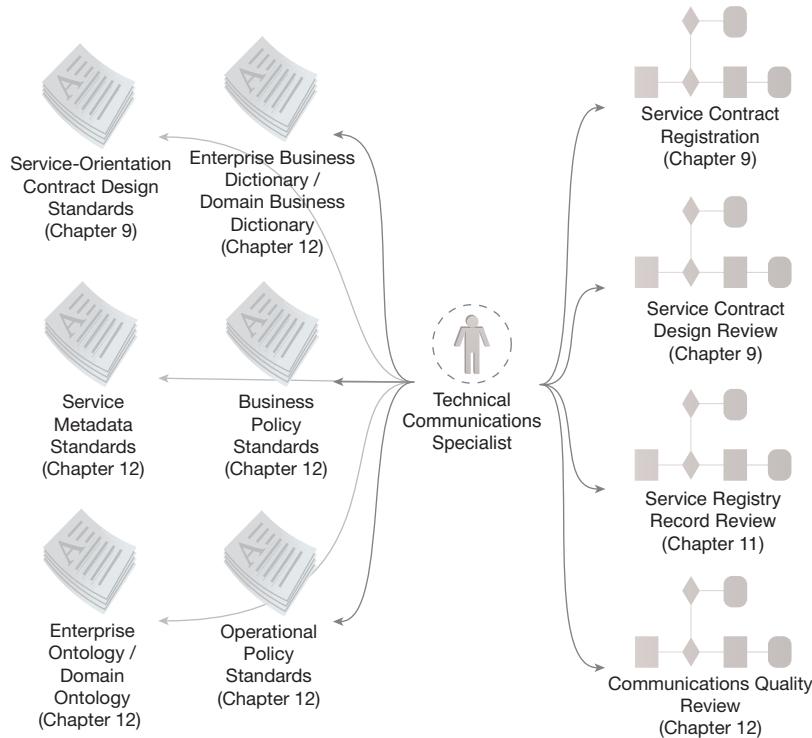


Figure B.10

SOA governance precepts and processes associated with the Technical Communications Specialist role.

Enterprise Architect

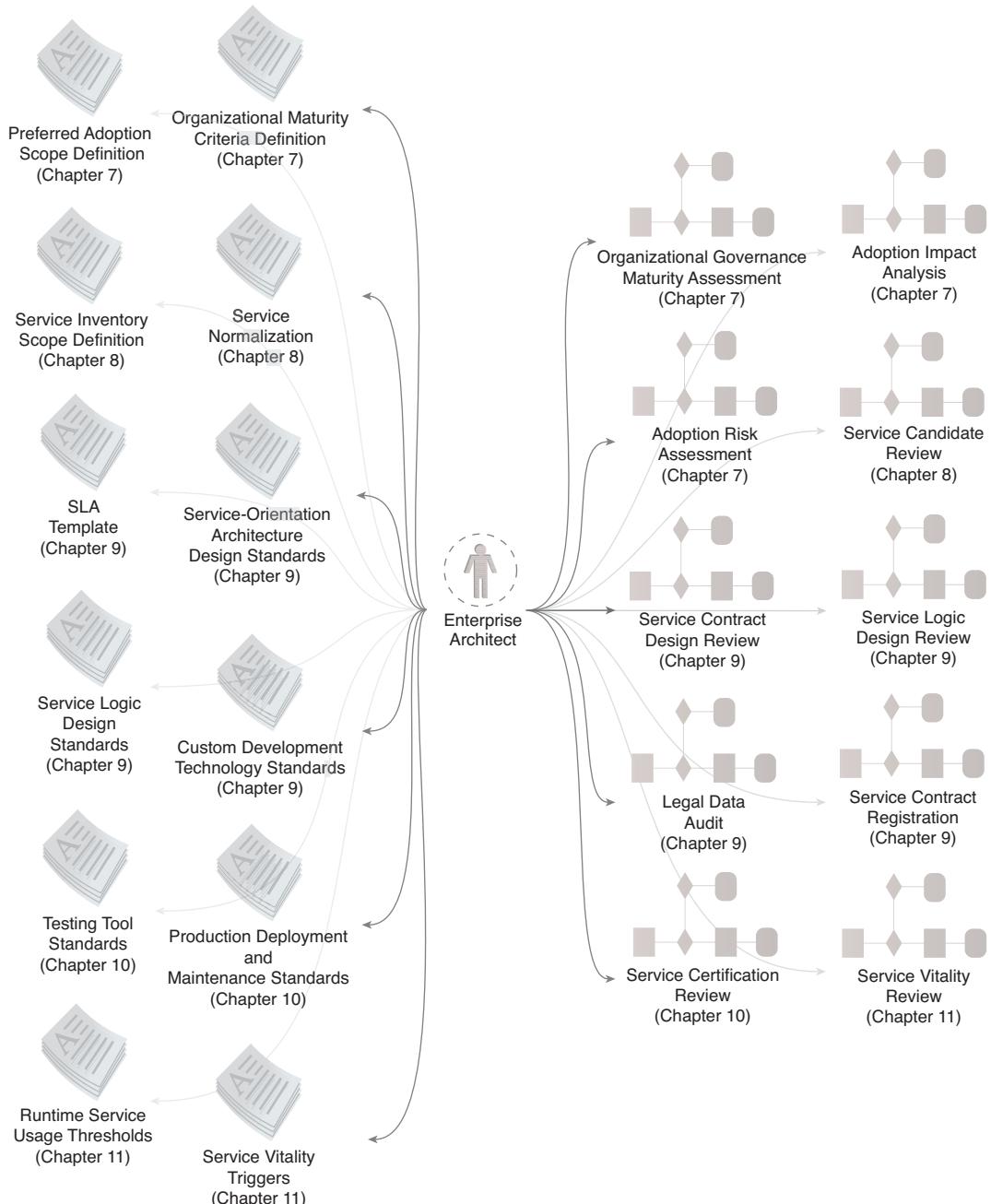


Figure B.11

SOA governance precepts and processes associated with the Enterprise Architect role.

Enterprise Design Standards Custodian (and Auditor)

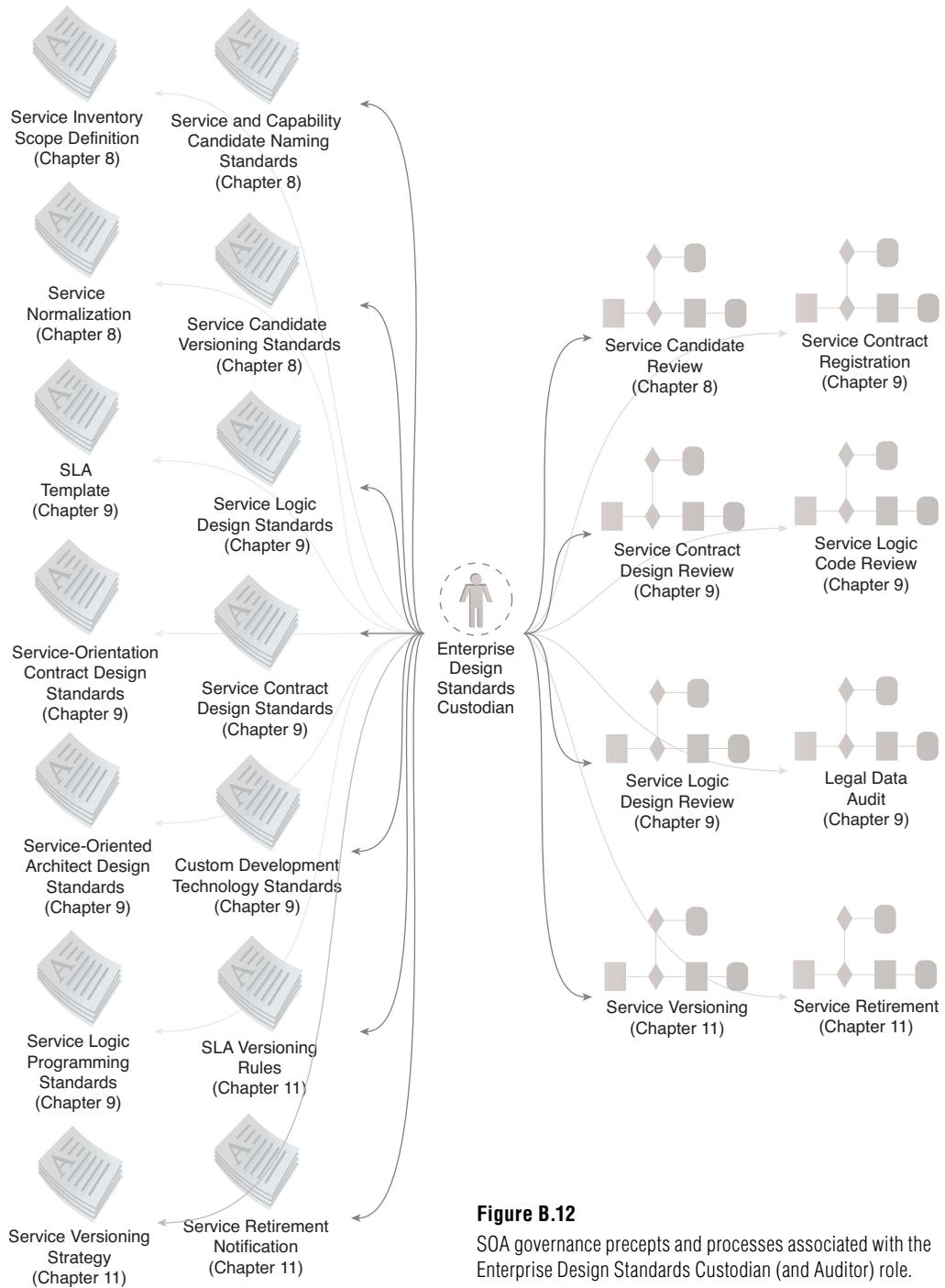


Figure B.12

SOA governance precepts and processes associated with the Enterprise Design Standards Custodian (and Auditor) role.

SOA Quality Assurance Specialist

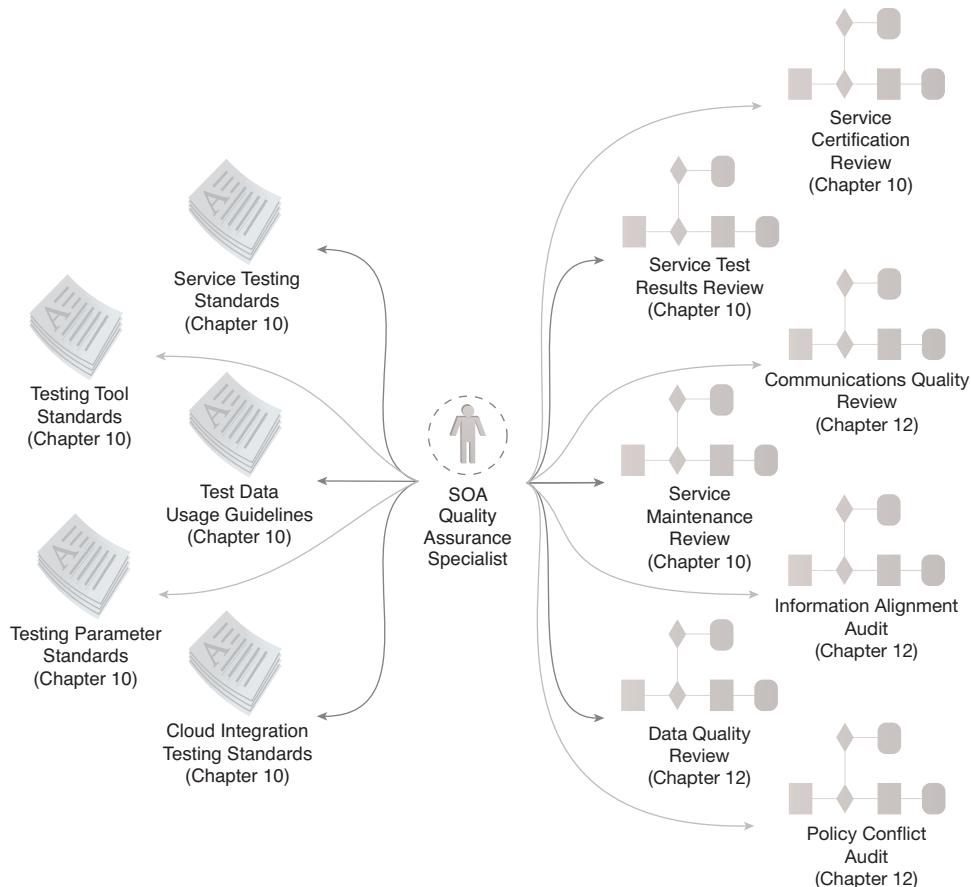


Figure B.13

SOA governance precepts and processes associated with the SOA Quality Assurance Specialist role.

SOA Security Specialist

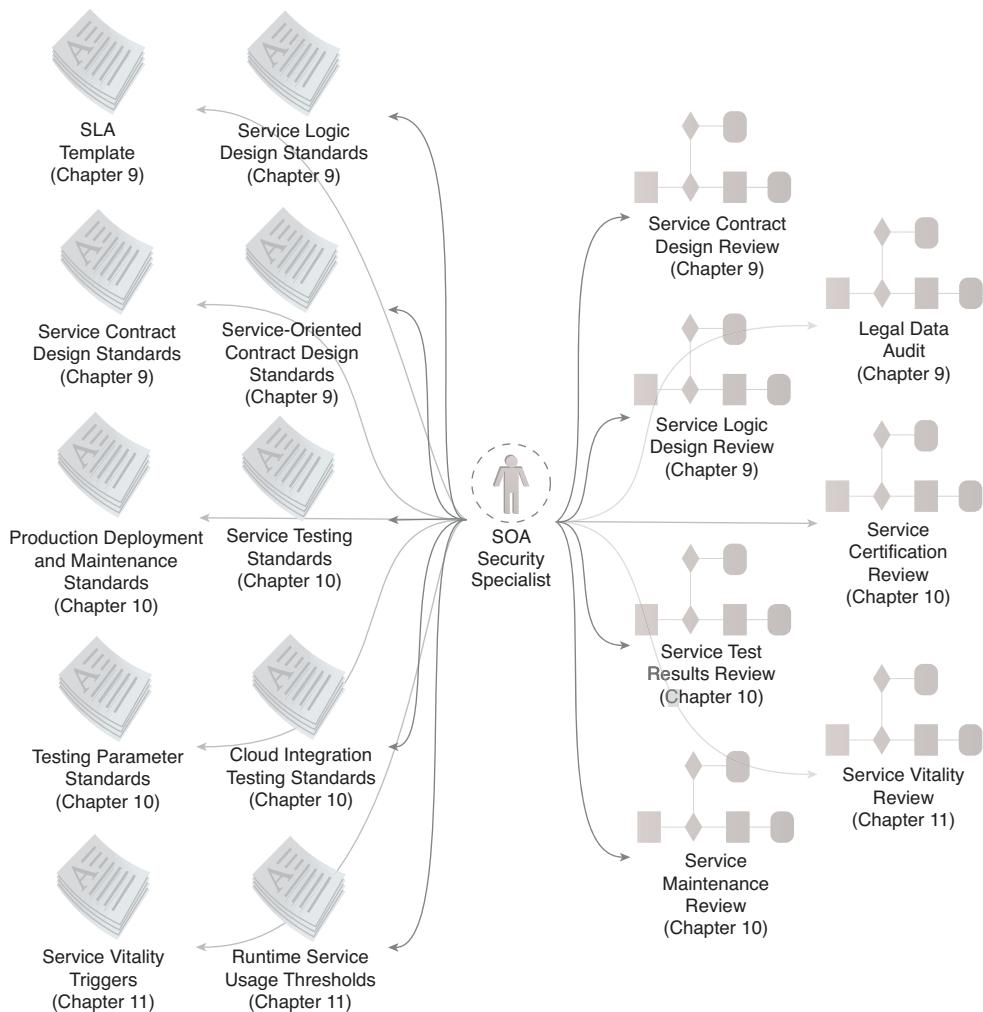


Figure B.14

SOA governance precepts and processes associated with the SOA Security Specialist role.

SOA Governance Specialist (precepts)



Figure B.15

SOA governance precepts associated with the SOA Governance Specialist role.

SOA Governance Specialist (processes)

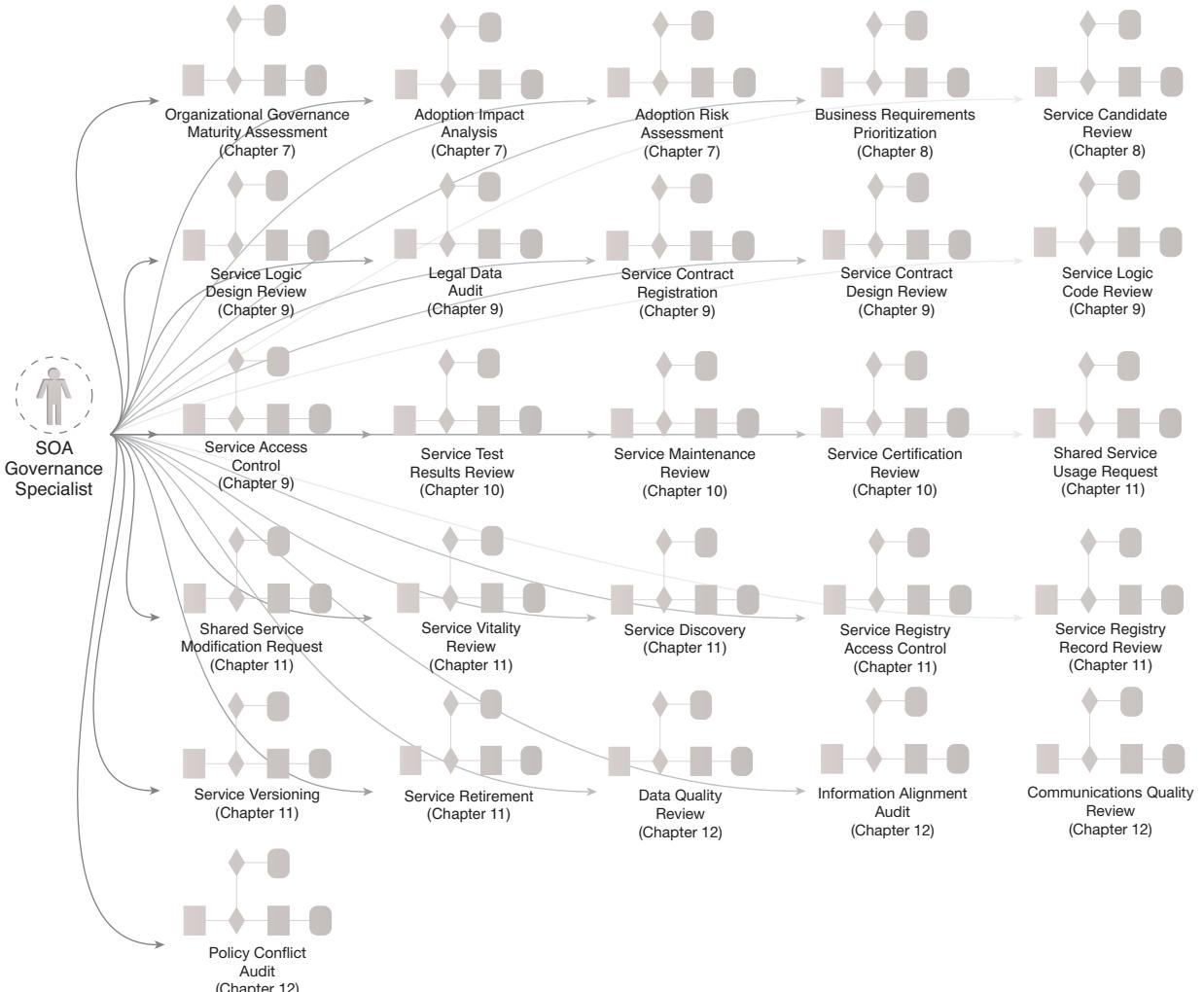
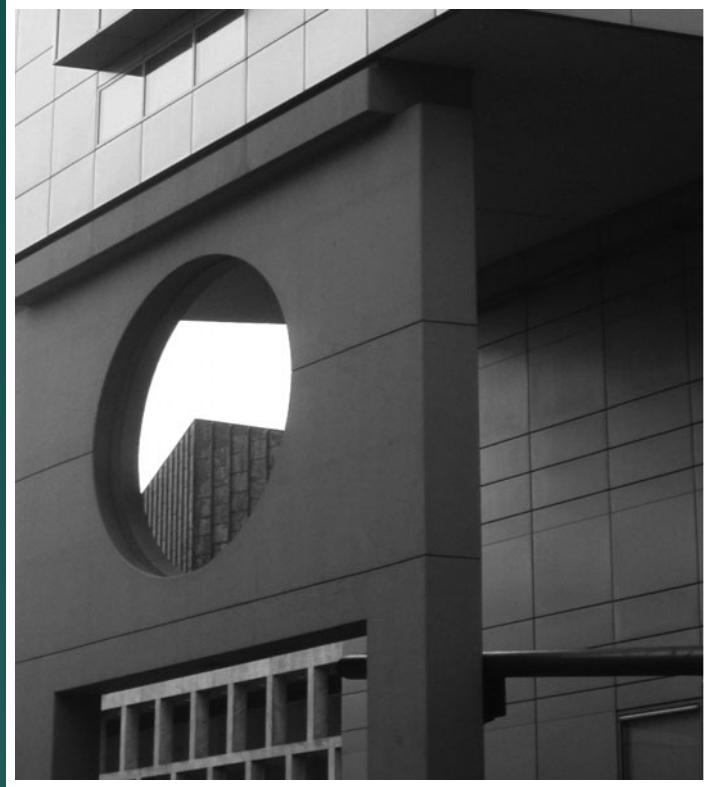


Figure B.16

SOA governance processes associated with the SOA Governance Specialist role.

Appendix C



Service-Orientation Principles Reference

This appendix provides profile tables for the eight design principles that are documented in *SOA Principles of Service Design*, a title that is part of this book series. Each principle that is referenced in this book is suffixed with the page number of its corresponding profile table in this appendix.

Every profile table contains the following sections:

- *Short Definition* – A concise, single-statement definition that establishes the fundamental purpose of the principle.
- *Long Definition* – A longer description of the principle that provides more detail as to what it is intended to accomplish.
- *Goals* – A list of specific design goals that are expected from the application of the principle. Essentially, this list provides the ultimate results of the principle's realization.
- *Design Characteristics* – A list of specific design characteristics that can be realized via the application of the principle. This provides some insight as to how the principle ends up shaping the service.
- *Implementation Requirements* – A list of common prerequisites for effectively applying the design principle. These can range from technology to organizational requirements.

Note that these tables provide only summarized content from the original publication. Information about service-orientation principles is also published online at www.soaprinciples.com.

Standardized Service Contract	
Short Definition	<i>"Services share standardized contracts."</i>
Long Definition	<i>"Services within the same service inventory are in compliance with the same contract design standards."</i>
Goals	<ul style="list-style-type: none">• To enable services with a meaningful level of natural interoperability within the boundary of a service inventory. This reduces the need for data transformation because consistent data models are used for information exchange.• To allow the purpose and capabilities of services to be more easily and intuitively understood. The consistency with which service functionality is expressed through service contracts increases interpretability and the overall predictability of service endpoints throughout a service inventory. <p>Note that these goals are further supported by other service-orientation principles as well.</p>
Design Characteristics	<ul style="list-style-type: none">• A service contract (comprised of a technical interface or one or more service description documents) is provided with the service.• The service contract is standardized through the application of design standards.
Implementation Requirements	<p>The fact that contracts need to be standardized can introduce significant implementation requirements to organizations that do not have a history of using standards.</p> <p>For example:</p> <ul style="list-style-type: none">• Design standards and conventions need to ideally be in place prior to the delivery of any service in order to ensure adequately scoped standardization. (For those organizations that have already produced ad-hoc Web services, retro-fitting strategies may need to be employed.)• Formal processes need to be introduced to ensure that services are modeled and designed consistently, incorporating accepted design principles, conventions, and standards.

- Because achieving standardized Web service contracts generally requires a “contract first” approach to service-oriented design, the full application of this principle will often demand the use of development tools capable of importing a customized service contract without imposing changes.
- Appropriate skill-sets are required to carry out the modeling and design processes with the chosen tools. When working with Web services, the need for a high level of proficiency with XML schema and WSDL languages is practically unavoidable. WS-Policy expertise may also be required.

These and other requirements can add up to a noticeable transition effort that goes well beyond technology adoption.

Table C.1

A profile for the Standardized Service Contract principle.

Service Loose Coupling	
Short Definition	<i>"Services are loosely coupled."</i>
Long Definition	<i>"Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment."</i>
Goals	By consistently fostering reduced coupling within and between services we are working toward a state where service contracts increase independence from their implementations and services are increasingly independent from each other. This promotes an environment in which services and their consumers can be adaptively evolved over time with minimal impact on each other.
Design Characteristics	<ul style="list-style-type: none">The existence of a service contract that is ideally decoupled from technology and implementation details.A functional service context that is not dependent on outside logic.Minimal consumer coupling requirements.
Implementation Requirements	<ul style="list-style-type: none">Loosely coupled services are typically required to perform more runtime processing than if they were more tightly coupled. As a result, data exchange in general can consume more runtime resources, especially during concurrent access and high usage scenarios.To achieve the right balance of coupling, while also supporting the other service-orientation principles that affect contract design, requires increased service contract design proficiency.

Table C.2

A profile for the Service Loose Coupling principle.

Service Abstraction	
Short Definition	<i>"Non-essential service information is abstracted."</i>
Long Definition	<i>"Service contracts only contain essential information and information about services is limited to what is published in service contracts."</i>
Goals	Many of the other principles emphasize the need to publish <i>more</i> information in the service contract. The primary role of this principle is to keep the quantity and detail of contract content concise and balanced and prevent unnecessary access to additional service details.
Design Characteristics	<ul style="list-style-type: none"> Services consistently abstract specific information about technology, logic, and function away from the outside world (the world outside of the service boundary). Services have contracts that concisely define interaction requirements and constraints and other required service meta details. Outside of what is documented in the service contract, information about a service is controlled or altogether hidden within a particular environment.
Implementation Requirements	The primary prerequisite to achieving the appropriate level of abstraction for each service is the level of service contract design skill applied.
Web Service Region of Influence	The <i>Region of Influence</i> part of this profile has been moved to the <i>Types of Meta Abstraction</i> section (in the book <i>SOA Principles of Service Design</i>) where a separate Web service figure is provided for each form of abstraction.

Table C.3

A profile for the Service Abstraction principle.

Service Reusability	
Short Definition	<i>"Services are reusable."</i>
Long Definition	<i>"Services contain and express agnostic logic and can be positioned as reusable enterprise resources."</i>
Goals	<p>The goals behind Service Reusability are tied directly to some of the most strategic objectives of service-oriented computing:</p> <ul style="list-style-type: none">• To allow for service logic to be repeatedly leveraged over time so as to achieve an increasingly high return on the initial investment of delivering the service.• To increase business agility on an organizational level by enabling the rapid fulfillment of future business automation requirements through wide-scale service composition.• To enable the realization of agnostic service models.• To enable the creation of service inventories with a high percentage of agnostic services.
Design Characteristics	<ul style="list-style-type: none">• <i>The service is defined by an agnostic functional context</i>—The logic encapsulated by the service is associated with a context that is sufficiently agnostic to any one usage scenario so as to be considered reusable.• <i>The service logic is highly generic</i>—The logic encapsulated by the service is sufficiently generic, allowing it to facilitate numerous usage scenarios by different types of service consumers.• <i>The service has a generic and extensible contract</i>—The service contract is flexible enough to process a range of input and output messages.• <i>The service logic can be accessed concurrently</i>—Services are designed to facilitate simultaneous access by multiple consumer programs.

Implementation Requirements	<p>From an implementation perspective, Service Reusability can be the most demanding of the principles we've covered so far. Below are common requirements for creating reusable services and supporting their long-term existence:</p> <ul style="list-style-type: none">• A scalable runtime hosting environment capable of high-to-extreme concurrent service usage. Once a service inventory is relatively mature, reusable services will find themselves in an increasingly large number of compositions.• A solid version control system to properly evolve contracts representing reusable services.• Service analysts and designers with a high degree of subject matter expertise who can ensure that the service boundary and contract accurately represent the service's reusable functional context.• A high level of service development and commercial software development expertise so as to structure the underlying logic into generic and potentially decomposable components and routines. <p>These and other requirements place an emphasis on the appropriate staffing of the service delivery team, as well as the importance of a powerful and scalable hosting environment and supporting infrastructure.</p>
------------------------------------	---

Table C.4

A profile for the Service Reusability principle.

Service Autonomy	
Short Definition	<i>"Services are autonomous."</i>
Long Definition	<i>"Services exercise a high level of control over their underlying runtime execution environment."</i>
Goals	<ul style="list-style-type: none">• To increase a service's runtime reliability, performance, and predictability, especially when being reused and composed.• To increase the amount of control a service has over its runtime environment. <p>By pursuing autonomous design and runtime environments, we are essentially aiming to increase post-implementation control over the service and the service's control over its own execution environment.</p>
Design Characteristics	<ul style="list-style-type: none">• Services have a contract that expresses a well-defined functional boundary that should not overlap with other services.• Services are deployed in an environment over which they exercise a great deal (and preferably an exclusive level) of control.• Service instances are hosted by an environment that accommodates high concurrency for scalability purposes.
Implementation Requirements	<ul style="list-style-type: none">• A high level of control over how service logic is designed and developed. Depending on the level of autonomy being sought, this may also involve control over the supporting data models.• A distributable deployment environment, so as to allow the service to be moved, isolated, or composed as required.• An infrastructure capable of supporting desired autonomy levels.

Table C.5

A profile for the Service Autonomy principle.

Service Statelessness	
Short Definition	<i>“Services minimize statefulness.”</i>
Long Definition	<i>“Services minimize resource consumption by deferring the management of state information when necessary.”</i>
Goals	<ul style="list-style-type: none"> • To increase service scalability. • To support the design of agnostic service logic and improve the potential for service reuse.
Design Characteristics	<p>What makes this somewhat of a unique principle is the fact that it is promoting a condition of the service that is temporary in nature. Depending on the service model and state deferral approach used, different types of design characteristics can be implemented. Some examples include:</p> <ul style="list-style-type: none"> • Highly business process-agnostic logic so that the service is not designed to retain state information for any specific parent business process. • Less constrained service contracts so as to allow for the receipt and transmission of a wider range of state data at runtime. • Increased amounts of interpretative programming routines capable of parsing a range of state information delivered by messages and responding to a range of corresponding action requests.
Implementation Requirements	<p>Although state deferral can reduce the overall consumption of memory and system resources, services designed with statelessness considerations can also introduce some performance demands associated with the runtime retrieval and interpretation of deferred state data.</p> <p>Here is a short checklist of common requirements that can be used to assess the support of stateless service designs by vendor technologies and target deployment locations:</p> <ul style="list-style-type: none"> • The runtime environment should allow for a service to transition from an idle state to an active processing state in a highly efficient manner.

- Enterprise-level or high-performance XML parsers and hardware accelerators (and SOAP processors) should be provided to allow services implemented as Web services to more efficiently parse larger message payloads with less performance constraints.
- The use of attachments may need to be supported by Web services to allow for messages to include bodies of payload data that do not undergo interface-level validation or translation to local formats.

The nature of the implementation support required by the average stateless service in an environment will depend on the state deferral approach used within the service-oriented architecture.

Table C.6

A profile for the Service Statelessness principle.

Service Discoverability	
Short Definition	<i>“Services are discoverable.”</i>
Long Definition	<i>“Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.”</i>
Goals	<ul style="list-style-type: none">• Services are positioned as highly discoverable resources within the enterprise.• The purpose and capabilities of each service are clearly expressed so that they can be interpreted by humans and software programs. <p>Achieving these goals requires foresight and a solid understanding of the nature of the service itself. Depending on the type of service model being designed, realizing this principle may require both business and technical expertise.</p>
Design Characteristics	<ul style="list-style-type: none">• Service contracts are equipped with appropriate meta data that will be correctly referenced when discovery queries are issued.• Service contracts are further outfitted with additional meta information that clearly communicates their purpose and capabilities to humans.• If a service registry exists, registry records are populated with the same attention to meta information as just described.• If a service registry does not exist, service profile documents are authored to supplement the service contract and to form the basis for future registry records. (See Chapter 15 in <i>SOA Principles of Service Design</i> for more details about service profiles.)

Implementation Requirements	<ul style="list-style-type: none">• The existence of design standards that govern the meta information used to make service contracts discoverable and interpretable, as well as guidelines for how and when service contracts should be further supplemented with annotations.• The existence of design standards that establish a consistent means of recording service meta information outside of the contract. This information is either collected in a supplemental document in preparation for a service registry, or it is placed in the registry itself. <p>You may have noticed the absence of a service registry on the list of implementation requirements. As previously established, the goal of this principle is to implement design characteristics within the service, not within the architecture.</p>
------------------------------------	---

Table C.7

A profile for the Service Discoverability principle.

Service Composability	
Short Definition	<i>"Services are composable."</i>
Long Definition	<i>"Services are effective composition participants, regardless of the size and complexity of the composition."</i>
Goals	<p>When discussing the goals of Service Composability, pretty much all of the goals of Service Reusability (479) apply. This is because service composition often turns out to be a form of service reuse. In fact, you may recall that one of the objectives we listed for the Service Reusability (479) principle was to enable wide-scale service composition.</p> <p>However, above and beyond simply attaining reuse, service composition provides the medium through which we can achieve what is often classified as the ultimate goal of service-oriented computing. By establishing an enterprise comprised of solution logic represented by an inventory of highly reusable services, we provide the means for a large extent of future business automation requirements to be fulfilled through ... you guessed it: service composition.</p>
Design Characteristics for Composition Member Capabilities	<p>Ideally, every service capability (especially those providing reusable logic) is considered a potential composition member. This essentially means that the design characteristics already established by the Service Reusability (479) principle are equally relevant to building effective composition members.</p> <p>Additionally, there are two further characteristics emphasized by this principle:</p> <ul style="list-style-type: none">• The service needs to possess a highly efficient execution environment. More so than being able to manage concurrency, the efficiency with which composition members perform their individual processing should be highly tuned.• The service contract needs to be flexible so that it can facilitate different types of data exchange requirements for similar functions. This typically relates to the ability of the contract to exchange the same type of data at different levels of granularity.

	<p>The manner in which these qualities go beyond mere reuse has to do primarily with the service being capable of optimizing its runtime processing responsibilities in support of multiple, simultaneous compositions.</p>
Design Characteristics for Composition Controller Capabilities	<p>Composition members will often also need to act as controllers or sub-controllers within different composition configurations. However, services designed as designated controllers are generally alleviated from many of the high-performance demands placed on composition members.</p> <p>These types of services therefore have their own set of design characteristics:</p> <ul style="list-style-type: none">• The logic encapsulated by a designated controller will almost always be limited to a single business task. Typically, the task service model is used, resulting in the common characteristics of that model being applied to this type of service.• While designated controllers may be reusable, service reuse is not usually a primary design consideration. Therefore, the design characteristics fostered by Service Reusability (479) are considered and applied where appropriate, but with less of the usual rigor applied to agnostic services.• Statelessness is not always as strictly emphasized on designated controllers as with composition members. Depending on the state deferral options available by the surrounding architecture, designated controllers may sometimes need to be designed to remain fully stateful while the underlying composition members carry out their respective parts of the overall task. <p>Of course, any capability acting as a controller can become a member of a larger composition, which brings the previously listed composition member design characteristics into account as well.</p>

Table C.8

A profile for the Service Composability principle.

This page intentionally left blank

Appendix D



SOA Design Patterns Reference

This appendix provides profile tables for all 85 patterns that are documented in *SOA Design Patterns*, a title that is part of this book series. Each pattern that is referenced in this book is suffixed with the page number of its corresponding profile table in this appendix.

Every profile table contains the following sections:

- *Requirement* – A requirement is a concise, single-sentence statement that presents the fundamental requirement addressed by the pattern in the form of a question. Every pattern description begins with this statement.
- *Icon* – Each pattern description is accompanied by an icon image that acts as a visual identifier. The icons are displayed together with the requirement statements in each pattern profile as well as on the inside book cover.
- *Problem* – The issue causing a problem and the effects of the problem. It is this problem for which the pattern is expected to provide a solution.
- *Solution* – This represents the design solution proposed by the pattern to solve the problem and fulfill the requirement.
- *Application* – This part is dedicated to describing how the pattern can be applied. It can include guidelines, implementation details, and sometimes even a suggested process.
- *Impacts* – This section highlights common consequences, costs, and requirements associated with the application of a pattern and may also provide alternatives that can be considered.
- *Principles* – References to related service-orientation principles.
- *Architecture* – References to related SOA architecture types (as described in Chapter 3).

Note that these tables provide only summarized content from the original publication. All pattern profile tables in this book are also published online at SOAPatterns.org.

Agnostic Capability

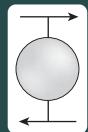
How can multi-purpose service logic be made effectively consumable and composable?



Problem	Service capabilities derived from specific concerns may not be useful to multiple service consumers, thereby reducing the reusability potential of the agnostic service.
Solution	Agnostic service logic is partitioned into a set of well-defined capabilities that address common concerns not specific to any one problem. Through subsequent analysis, the agnostic context of capabilities is further refined.
Application	Service capabilities are defined and iteratively refined through proven analysis and modeling processes.
Impacts	The definition of each service capability requires extra up-front analysis and design effort.
Principles	Standardized Service Contract (475), Service Reusability (479), Service Composability (486)
Architecture	Service

Agnostic Context

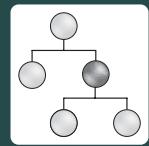
How can multi-purpose service logic be positioned as an effective enterprise resource?



Problem	Multi-purpose logic grouped together with single purpose logic results in programs with little or no reuse potential that introduce waste and redundancy into an enterprise.
Solution	Isolate logic that is not specific to one purpose into separate services with distinct agnostic contexts.
Application	Agnostic service contexts are defined by carrying out service-oriented analysis and service modeling processes.
Impacts	This pattern positions reusable solution logic at an enterprise level, potentially bringing with it increased design complexity and enterprise governance issues.
Principles	Service Reusability (479)
Architecture	Service

Agnostic Sub-Controller

How can agnostic, cross-entity composition logic be separated, reused, and governed independently?



Problem	Service compositions are generally configured specific to a parent task, inhibiting reuse potential that may exist within a subset of the composition logic.
Solution	Reusable, cross-entity composition logic is abstracted or made accessible via an agnostic sub-controller capability, allowing that subset of the parent composition logic to be recomposed independently.
Application	A new agnostic service is created or a task service is appended with an agnostic sub-controller capability.
Impacts	The addition of a cross-entity, agnostic service can increase the size and complexity of compositions and the abstraction of agnostic cross-entity logic can violate modeling and design standards established by Service Layers [S61].
Principles	Service Reusability (479), Service Composability (486)
Architecture	Composition, Service

Asynchronous Queuing

By Mark Little, Thomas Rischbeck, Arnaud Simon

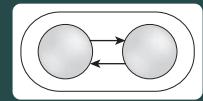


How can a service and its consumers accommodate isolated failures and avoid unnecessarily locking resources?

Problem	When a service capability requires that consumers interact with it synchronously, it can inhibit performance and compromise reliability.
Solution	A service can exchange messages with its consumers via an intermediary buffer, allowing service and consumers to process messages independently by remaining temporally decoupled.
Application	Queuing technology needs to be incorporated into the surrounding architecture, and back-up stores may also be required.
Impacts	There may be no acknowledgement of successful message delivery, and atomic transactions may not be possible.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Statelessness (482)
Architecture	Inventory, Composition

Atomic Service Transaction

How can a transaction with rollback capability be propagated across messaging-based services?



Problem	When runtime activities that span multiple services fail, the parent business task is incomplete and actions performed and changes made up to that point may compromise the integrity of the underlying solution and architecture.
Solution	Runtime service activities can be wrapped in a transaction with rollback feature that resets all actions and changes if the parent business task cannot be successfully completed.
Application	A transaction management system is made part of the inventory architecture and then used by those service compositions that require rollback features.
Impacts	Transacted service activities can consume more memory because of the requirement for each service to preserve its original state until it is notified to rollback or commit its changes.
Principles	Service Statelessness (482)
Architecture	Inventory, Composition

Brokered Authentication

By Jason Hogg, Don Smith, Fred Chong, Tom Hollander, Wojtek Kozaczynski, Larry Brader, Nelly Delgado, Dwayne Taylor, Lonnie Wall, Paul Slater, Sajjad Nasir Imran, Pablo Cibraro, Ward Cunningham



How can a service efficiently verify consumer credentials if the consumer and service do not trust each other or if the consumer requires access to multiple services?

Problem	Requiring the use of Direct Authentication [518] can be impractical or even impossible when consumers and services do not trust each other or when consumers are required to access multiple services as part of the same runtime activity.
Solution	An authentication broker with a centralized identity store assumes the responsibility for authenticating the consumer and issuing a token that the consumer can use to access the service.
Application	An authentication broker product introduced into the inventory architecture carries out the intermediary authentication and issuance of temporary credentials using technologies such as X.509 certificates or Kerberos, SAML, or SecPAL tokens.
Impacts	This pattern can establish a potential single point of failure and a central breach point that, if compromised, could jeopardize an entire service inventory.
Principles	Service Composability (486)
Architecture	Inventory, Composition, Service

Canonical Expression

How can service contracts be consistently understood and interpreted?



Problem	Service contracts may express similar capabilities in different ways, leading to inconsistency and risking misinterpretation.
Solution	Service contracts are standardized using naming conventions.
Application	Naming conventions are applied to service contracts as part of formal analysis and design processes.
Impacts	The use of global naming conventions introduces enterprise-wide standards that need to be consistently used and enforced.
Principles	Standardized Service Contract (475), Service Discoverability (484)
Architecture	Enterprise, Inventory, Service

Canonical Protocol	
<i>How can services be designed to avoid protocol bridging?</i>	
Problem	Services that support different communication technologies compromise interoperability, limit the quantity of potential consumers, and introduce the need for undesirable protocol bridging measures.
Solution	The architecture establishes a single communications technology as the sole or primary medium by which services can interact.
Application	The communication protocols (including protocol versions) used within a service inventory boundary are standardized for all services.
Impacts	An inventory architecture in which communication protocols are standardized is subject to any limitations imposed by the communications technology.
Principles	Standardized Service Contract (475)
Architecture	Inventory, Service



Canonical Resources

How can unnecessary infrastructure resource disparity be avoided?



Problem	Service implementations can unnecessarily introduce disparate infrastructure resources, thereby bloating the enterprise and resulting in increased governance burden.
Solution	The supporting infrastructure and architecture can be equipped with common resources and extensions that can be repeatedly utilized by different services.
Application	Enterprise design standards are defined to formalize the required use of standardized architectural resources.
Impacts	If this pattern leads to too much dependency on shared infrastructure resources, it can decrease the autonomy and mobility of services.
Principles	Service Autonomy (481)
Architecture	Enterprise, Inventory

Canonical Schema

How can services be designed to avoid data model transformation?



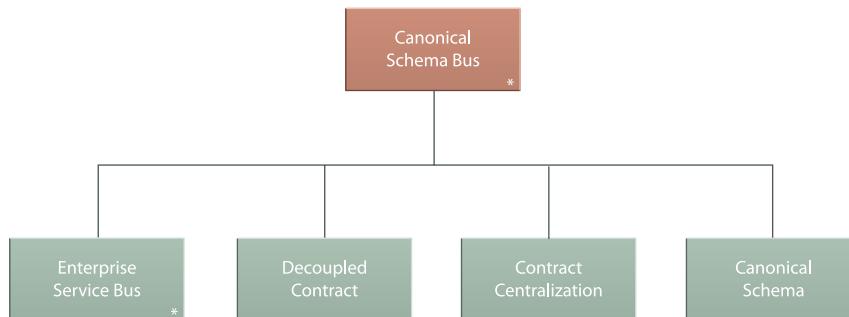
Problem	Services with disparate models for similar data impose transformation requirements that increase development effort, design complexity, and runtime performance overhead.
Solution	Data models for common information sets are standardized across service contracts within an inventory boundary.
Application	Design standards are applied to schemas used by service contracts as part of a formal design process.
Impacts	Maintaining the standardization of contract schemas can introduce significant governance effort and cultural challenges.
Principles	Standardized Service Contract (475)
Architecture	Inventory, Service

Canonical Schema Bus

By Clemens Utschig-Utschig, Berthold Maier, Bernd Trops, Hajo Normann, Torsten Winterberg, Thomas Erl

While Enterprise Service Bus [523] provides a range of messaging-centric functions that help establish connectivity between different services and between services and resources they are required to encapsulate, it does not inherently enforce or advocate standardization.

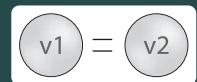
Building upon the platform established by Enterprise Service Bus [523], this pattern positions entry points into the logic, data, and functions offered via the service bus environment as independently standardized service contracts.



Canonical Schema Bus is comprised of the co-existent application of Enterprise Service Bus [523], Decoupled Contract [517], Contract Centralization [509], and Canonical Schema [500].

Canonical Versioning

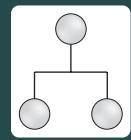
How can service contracts within the same service inventory be versioned with minimal impact?



Problem	Service contracts within the same service inventory that are versioned differently will cause numerous interoperability and governance problems.
Solution	Service contract versioning rules and the expression of version information are standardized within a service inventory boundary.
Application	Governance and design standards are required to ensure consistent versioning of service contracts within the inventory boundary.
Impacts	The creation and enforcement of the required versioning standards introduce new governance demands.
Principles	Standardized Service Contract (475)
Architecture	Service, Inventory

Capability Composition

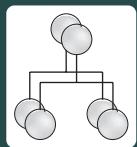
How can a service capability solve a problem that requires logic outside of the service boundary?



Problem	A capability may not be able to fulfill its processing requirements without adding logic that resides outside of its service's functional context, thereby compromising the integrity of the service context and risking service denormalization.
Solution	When requiring access to logic that falls outside of a service's boundary, capability logic within the service is designed to compose one or more capabilities in other services.
Application	The functionality encapsulated by a capability includes logic that can invoke other capabilities from other services.
Impacts	Carrying out composition logic requires external invocation, which adds performance overhead and decreases service autonomy.
Principles	All
Architecture	Inventory, Composition, Service

Capability Recomposition

How can the same capability be used to help solve multiple problems?

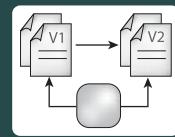


Problem	Using agnostic service logic to only solve a single problem is wasteful and does not leverage the logic's reuse potential.
Solution	Agnostic service capabilities can be designed to be repeatedly invoked in support of multiple compositions that solve multiple problems.
Application	Effective recomposition requires the coordinated, successful, and repeated application of several additional patterns.
Impacts	Repeated service composition demands existing and persistent standardization and governance.
Principles	All
Architecture	Inventory, Composition, Service

Compatible Change

By David Orchard, Chris Riley

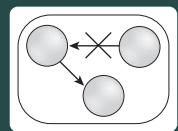
How can a service contract be modified without impacting consumers?



Problem	Changing an already-published service contract can impact and invalidate existing consumer programs.
Solution	Some changes to the service contract can be backwards-compatible, thereby avoiding negative consumer impacts.
Application	Service contract changes can be accommodated via extension or by the loosening of existing constraints or by applying Concurrent Contracts [508].
Impacts	Compatible changes still introduce versioning governance effort, and the technique of loosening constraints can lead to vague contract designs.
Principles	Standardized Service Contract (475), Service Loose Coupling (477)
Architecture	Service

Compensating Service Transaction

By Clemens Utschig-Utschig, Berthold Maier, Bernd Trops, Hajo Normann, Torsten Winterberg, Brian Loesgen, Mark Little

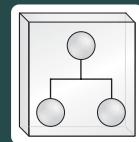


How can composition runtime exceptions be consistently accommodated without requiring services to lock resources?

Problem	Whereas uncontrolled runtime exceptions can jeopardize a service composition, wrapping the composition in an atomic transaction can tie up too many resources, thereby negatively affecting performance and scalability.
Solution	Compensating routines are introduced, allowing runtime exceptions to be resolved with the opportunity for reduced resource locking and memory consumption.
Application	Compensation logic is pre-defined and implemented as part of the parent composition controller logic or via individual “undo” service capabilities.
Impacts	Unlike atomic transactions that are governed by specific rules, the use of compensation logic is open-ended and can vary in its actual effectiveness.
Principles	Service Loose Coupling (477)
Architecture	Inventory, Composition

Composition Autonomy

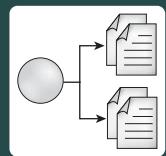
How can compositions be implemented to minimize loss of autonomy?



Problem	Composition controller services naturally lose autonomy when delegating processing tasks to composed services, some of which may be shared across multiple compositions.
Solution	All composition participants can be isolated to maximize the autonomy of the composition as a whole.
Application	The agnostic member services of a composition are redundantly implemented in an isolated environment together with the task service.
Impacts	Increasing autonomy on a composition level results in increased infrastructure costs and government responsibilities.
Principles	Service Autonomy (481), Service Reusability (479), Service Composability (486)
Architecture	Composition

Concurrent Contracts

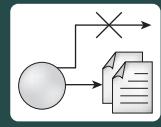
How can a service facilitate multi-consumer coupling requirements and abstraction concerns at the same time?



Problem	A service's contract may not be suitable for or applicable to all potential service consumers.
Solution	Multiple contracts can be created for a single service, each targeted at a specific type of consumer.
Application	This pattern is ideally applied together with Service Façade [558] to support new contracts as required.
Impacts	Each new contract can effectively add a new service endpoint to an inventory, thereby increasing corresponding governance effort.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Reusability (479)
Architecture	Service

Contract Centralization

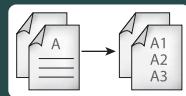
How can direct consumer-to-implementation coupling be avoided?



Problem	Consumer programs can be designed to access underlying service resources using different entry points, resulting in different forms of implementation dependencies that inhibit the service from evolving in response to change.
Solution	Access to service logic is limited to the service contract, forcing consumers to avoid implementation coupling.
Application	This pattern is realized through formal enterprise design standards and the targeted application of the Service Abstraction (478) design principle.
Impacts	Forcing consumer programs to access service capabilities and resources via a central contract can impose performance overhead and requires on-going standardization effort.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Abstraction (478)
Architecture	Composition, Service

Contract Denormalization

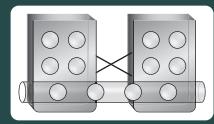
How can a service contract facilitate consumer programs with differing data exchange requirements?



Problem	Services with strictly normalized contracts can impose unnecessary functional and performance demands on some consumer programs.
Solution	Service contracts can include a measured extent of denormalization, allowing multiple capabilities to redundantly express core functions in different ways for different types of consumer programs.
Application	The service contract is carefully extended with additional capabilities that provide functional variations of a primary capability.
Impacts	Overuse of this pattern on the same contract can dramatically increase its size, making it difficult to interpret and unwieldy to govern.
Principles	Standardized Service Contract (475), Service Loose Coupling (477)
Architecture	Service

Cross-Domain Utility Layer

How can redundant utility logic be avoided across domain service inventories?



Problem	While domain service inventories may be required for independent business governance, they can impose unnecessary redundancy within utility service layers.
Solution	A common utility service layer can be established, spanning two or more domain service inventories.
Application	A common set of utility services needs to be defined and standardized in coordination with service inventory owners.
Impacts	Increased effort is required to coordinate and govern a cross-inventory utility service layer.
Principles	Service Reusability (479), Service Composability (486)
Architecture	Enterprise, Inventory

Data Confidentiality

By Jason Hogg, Don Smith, Fred Chong, Tom Hollander, Wojtek Kozaczynski, Larry Brader, Nelly Delgado, Dwayne Taylor, Lonnie Wall, Paul Slater, Sajjad Nasir Imran, Pablo Cibraro, Ward Cunningham



How can data within a message be protected so that it is not disclosed to unintended recipients while in transit?

Problem	Within service compositions, data is often required to pass through one or more intermediaries. Point-to-point security protocols, such as those frequently used at the transport-layer, may allow messages containing sensitive information to be intercepted and viewed by such intermediaries.
Solution	The message contents are encrypted independently from the transport, ensuring that only intended recipients can access the protected data.
Application	A symmetric or asymmetric encryption and decryption algorithm, such as those specified in the XML-Encryption standard, is applied at the message level.
Impacts	This pattern may add runtime performance overhead associated with the required encryption and decryption of message data. The management of keys can further add to governance burden.
Principles	Service Composability (486)
Architecture	Inventory, Composition, Service

Data Format Transformation

By Mark Little, Thomas Rischbeck, Arnaud Simon



How can services interact with programs that communicate with different data formats?

Problem	A service may be incompatible with resources it needs to access due to data format disparity. Furthermore, a service consumer that communicates using a data format different from a target service will be incompatible and therefore unable to invoke the service.
Solution	Intermediary data format transformation logic needs to be introduced in order to dynamically translate one data format into another.
Application	This necessary transformation logic is incorporated by adding internal service logic, service agents, or a dedicated transformation service.
Impacts	The use of data format transformation logic inevitably adds development effort, design complexity, and performance overhead.
Principles	Standardized Service Contract (475), Service Loose Coupling (477)
Architecture	Inventory, Composition, Service

Data Model Transformation

How can services interoperate when using different data models for the same type of data?



Problem	Services may use incompatible schemas to represent the same data, hindering service interaction and composition.
Solution	A data transformation technology can be incorporated to convert data between disparate schema structures.
Application	Mapping logic needs to be developed and deployed so that data compliant to one data model can be dynamically converted to comply to a different data model.
Impacts	Data model transformation introduces development effort, design complexity, and runtime performance overhead, and overuse of this pattern can seriously inhibit service recomposition potential.
Principles	Standardized Service Contract (475), Service Reusability (479), Service Composability (486)
Architecture	Inventory, Composition

Data Origin Authentication

By Jason Hogg, Don Smith, Fred Chong, Tom Hollander, Wojtek Kozaczynski, Larry Brader, Nelly Delgado, Dwayne Taylor, Lonnie Wall, Paul Slater, Sajjad Nasir Imran, Pablo Cibraro, Ward Cunningham

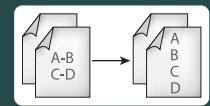


How can a service verify that a message originates from a known sender and that the message has not been tampered with in transit?

Problem	The intermediary processing layers generally required by service compositions can expose sensitive data when security is limited to point-to-point protocols, such as those used with transport-layer security.
Solution	A message can be digitally signed so that the recipient services can verify that it originated from the expected consumer and that it has not been tampered with during transit.
Application	A digital signature algorithm is applied to the message to provide “proof of origin,” allowing sensitive message contents to be protected from tampering. This technology must be supported by both consumer and service.
Impacts	Use of cryptographic techniques can add to performance requirements and the choice of digital signing algorithm can affect the level of security actually achieved.
Principles	Service Composability (486)
Architecture	Composition

Decomposed Capability

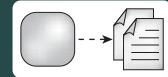
How can a service be designed to minimize the chances of capability logic deconstruction?



Problem	The decomposition of a service subsequent to its implementation can require the deconstruction of logic within capabilities, which can be disruptive and make the preservation of a service contract problematic.
Solution	Services prone to future decomposition can be equipped with a series of granular capabilities that more easily facilitate decomposition.
Application	Additional service modeling is carried out to define granular, more easily distributed capabilities.
Impacts	Until the service is eventually decomposed, it may be represented by a bloated contract that stays with it as long as proxy capabilities are supported.
Principles	Standardized Service Contract (475), Service Abstraction (478)
Architecture	Service

Decoupled Contract

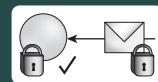
How can a service express its capabilities independently of its implementation?



Problem	For a service to be positioned as an effective enterprise resource, it must be equipped with a technical contract that exists independently from its implementation yet still in alignment with other services.
Solution	The service contract is physically decoupled from its implementation.
Application	A service's technical interface is physically separated and subject to relevant service-orientation design principles.
Impacts	Service functionality is limited to the feature-set of the decoupled contract medium.
Principles	Standardized Service Contract (475), Service Loose Coupling (477)
Architecture	Service

Direct Authentication

By Jason Hogg, Don Smith, Fred Chong, Tom Hollander, Wojtek Kozaczynski, Larry Brader, Nelly Delgado, Dwayne Taylor, Lonnie Wall, Paul Slater, Sajjad Nasir Imran, Pablo Cibraro, Ward Cunningham

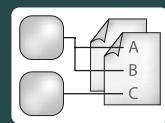


How can a service verify the credentials provided by a consumer?

Problem	Some of the capabilities offered by a service may be intended for specific groups of consumers or may involve the transmission of sensitive data. Attackers that access this data could use it to compromise the service or the IT enterprise itself.
Solution	Service capabilities require that consumers provide credentials that can be authenticated against an identity store.
Application	The service implementation is provided access to an identity store, allowing it to authenticate the consumer directly.
Impacts	Consumers must provide credentials compatible with the service's authentication logic. This pattern may lead to multiple identity stores, resulting in extra governance burden.
Principles	Service Composability (486)
Architecture	Composition, Service

Distributed Capability

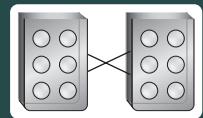
How can a service preserve its functional context while also fulfilling special capability processing requirements?



Problem	A capability that belongs within a service may have unique processing requirements that cannot be accommodated by the default service implementation, but separating capability logic from the service will compromise the integrity of the service context.
Solution	The underlying service logic is distributed, thereby allowing the implementation logic for a capability with unique processing requirements to be physically separated, while continuing to be represented by the same service contract.
Application	The logic is moved and intermediary processing is added to act as a liaison between the moved logic and the main service logic.
Impacts	The distribution of a capability's logic leads to performance overhead associated with remote communication and the need for new intermediate processing.
Principles	Standardized Service Contract (475), Service Autonomy (481)
Architecture	Service

Domain Inventory

How can services be delivered to maximize recombination when enterprise-wide standardization is not possible?



Problem	Establishing a single enterprise service inventory may be unmanageable for some enterprises, and attempts to do so may jeopardize the success of an SOA adoption as a whole.
Solution	Services can be grouped into manageable, domain-specific service inventories, each of which can be independently standardized, governed, and owned.
Application	Inventory domain boundaries need to be carefully established.
Impacts	Standardization disparity between domain service inventories imposes transformation requirements and reduces the overall benefit potential of the SOA adoption.
Principles	Standardized Service Contract (475), Service Abstraction (478), Service Composability (486)
Architecture	Enterprise, Inventory

Dual Protocols

How can a service inventory overcome the limitations of its canonical protocol while still remaining standardized?



Problem	Canonical Protocol [498] requires that all services conform to the use of the same communications technology; however, a single protocol may not be able to accommodate all service requirements, thereby introducing limitations.
Solution	The service inventory architecture is designed to support services based on primary and secondary protocols.
Application	Primary and secondary service levels are created and collectively represent the service endpoint layer. All services are subject to standard service-orientation design considerations and specific guidelines are followed to minimize the impact of not following Canonical Protocol [498].
Impacts	This pattern can lead to a convoluted inventory architecture, increased governance effort and expense, and (when poorly applied) an unhealthy dependence on Protocol Bridging [546]. Because the endpoint layer is semi-federated, the quantity of potential consumers and reuse opportunities is decreased.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Abstraction (478), Service Autonomy (481), Service Composability (486)
Architecture	Inventory, Service

Enterprise Inventory

How can services be delivered to maximize recomposition?

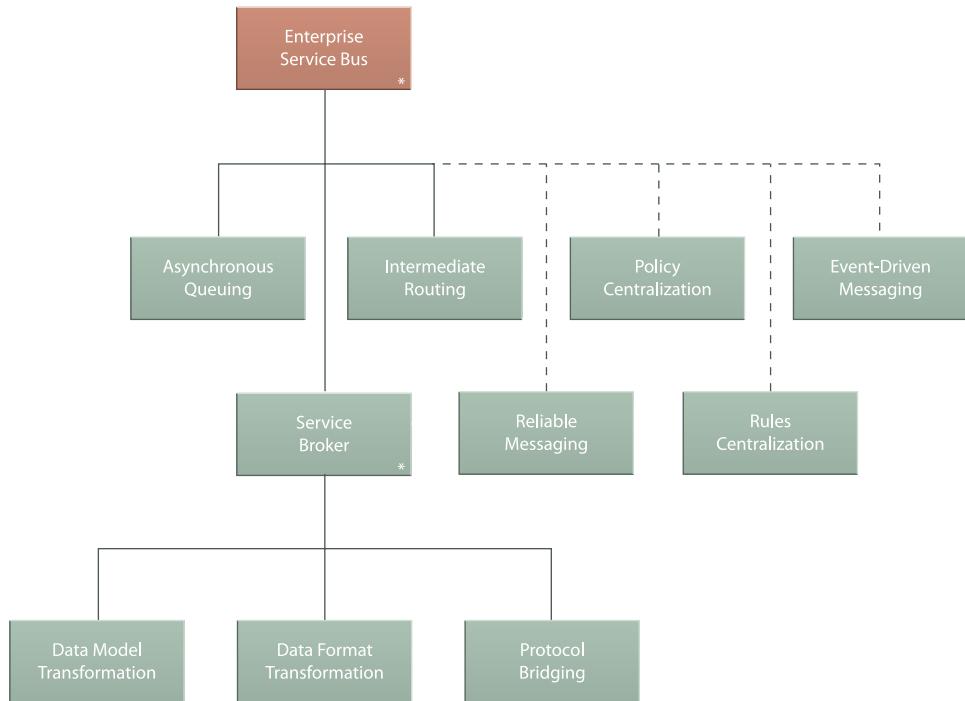


Problem	Delivering services independently via different project teams across an enterprise establishes a constant risk of producing inconsistent service and architecture implementations, compromising recomposition opportunities.
Solution	Services for multiple solutions can be designed for delivery within a standardized, enterprise-wide inventory architecture wherein they can be freely and repeatedly recomposed.
Application	The enterprise service inventory is ideally modeled in advance, and enterprise-wide standards are applied to services delivered by different project teams.
Impacts	Significant upfront analysis is required to define an enterprise inventory blueprint and numerous organizational impacts result from the subsequent governance requirements.
Principles	Standardized Service Contract (475), Service Abstraction (478), Service Composability (486)
Architecture	Enterprise, Inventory

Enterprise Service Bus

By Thomas Erl, Mark Little, Thomas Rischbeck, Arnaud Simon

An enterprise service bus represents an environment designed to foster sophisticated interconnectivity between services. It establishes an intermediate layer of processing that can help overcome common problems associated with reliability, scalability, and communications disparity.



Enterprise Service Bus is fundamentally comprised of the co-existent application of Asynchronous Queuing [494], Intermediate Routing [530], and Service Broker [553], and can be further extended via Reliable Messaging [549], Policy Centralization [543], Rules Centralization [550], and Event-Driven Messaging [525].

Entity Abstraction

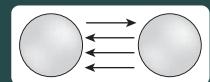
How can agnostic business logic be separated, reused, and governed independently?



Problem	Bundling both process-agnostic and process-specific business logic into the same service eventually results in the creation of redundant agnostic business logic across multiple services.
Solution	An agnostic business service layer can be established, dedicated to services that base their functional context on existing business entities.
Application	Entity service contexts are derived from business entity models and then establish a logical layer that is modeled during the analysis phase.
Impacts	The core, business-centric nature of the services introduced by this pattern require extra modeling and design attention and their governance requirements can impose dramatic organizational changes.
Principles	Service Loose Coupling (477), Service Abstraction (478), Service Reusability (479), Service Composability (486)
Architecture	Inventory, Composition, Service

Event-Driven Messaging

By Mark Little, Thomas Rischbeck, Arnaud Simon

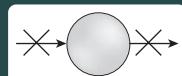


How can service consumers be automatically notified of runtime service events?

Problem	Events that occur within the functional boundary encapsulated by a service may be of relevance to service consumers, but without resorting to inefficient polling-based interaction, the consumer has no way of learning about these events.
Solution	The consumer establishes itself as a subscriber of the service. The service, in turn, automatically issues notifications of relevant events to this and any of its subscribers.
Application	A messaging framework is implemented capable of supporting the publish-and-subscribe MEP and associated complex event processing and tracking.
Impacts	Event-driven message exchanges cannot easily be incorporated as part of Atomic Service Transaction [495], and publisher/subscriber availability issues can arise.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Autonomy (481)
Architecture	Inventory, Composition

Exception Shielding

By Jason Hogg, Don Smith, Fred Chong, Tom Hollander, Wojtek Kozaczynski, Larry Brader, Nelly Delgado, Dwayne Taylor, Lonnie Wall, Paul Slater, Sajjad Nasir Imran, Pablo Cibraro, Ward Cunningham



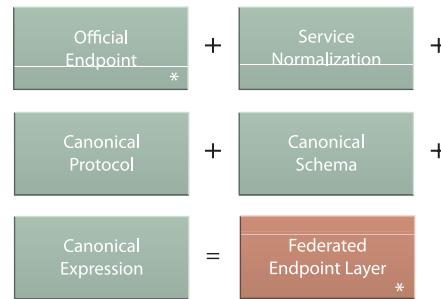
How can a service prevent the disclosure of information about its internal implementation when an exception occurs?

Problem	Unfiltered exception data output by a service may contain internal implementation details that can compromise the security of the service and its surrounding environment.
Solution	Potentially unsafe exception data is “sanitized” by replacing it with exception data that is safe by design before it is made available to consumers.
Application	This pattern can be applied at design time by reviewing and altering source code or at runtime by adding dynamic sanitization routines.
Impacts	Sanitized exception information can make the tracking of errors more difficult due to the lack of detail provided to consumers.
Principles	Service Abstraction (478)
Architecture	Service

Federated Endpoint Layer

Federation is an important concept in service-oriented computing. It represents the desired state of the external, consumer-facing perspective of a service inventory, as expressed by the collective contracts of all the inventory's services.

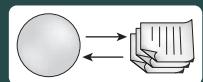
The more federated and unified this collection of contracts (endpoints) is, the more easily and effectively the services can be repeatedly consumed and leveraged.



The joint application of Official Endpoint [539], Service Normalization [563], Canonical Protocol [498], Canonical Schema [500], and Canonical Expression [497] results in Federated Endpoint Layer.

File Gateway

By Satadru Roy

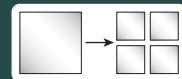


How can service logic interact with legacy systems that can only share information by exchanging files?

Problem	Data records contained in flat files produced by a legacy system need to be processed individually by service logic, but legacy systems are not capable of directly invoking services. Conversely, service logic may need to produce information for the legacy system, but building file creation and transfer functionality into the service can result in an inflexible design.
Solution	Intermediary two-way file processing logic is positioned between the legacy system and the service.
Application	For inbound data the file gateway processing logic can detect file drops and leverage available broker features to perform Data Model Transformation [514] and Data Format Transformation [513]. On the outbound side, this logic intercepts information produced by services and packages them (with possible transformation) into new or existing files for consumption by the legacy system.
Impacts	The type of logic provided by this pattern is unsuitable when immediate replies are required by either service or legacy system. Deployment and governance of two-way file processing logic can further add to operational complexity and may require specialized administration skills.
Principles	Service Loose Coupling (477)
Architecture	Service

Functional Decomposition

How can a large business problem be solved without having to build a standalone body of solution logic?



Problem	To solve a large, complex business problem a corresponding amount of solution logic needs to be created, resulting in a self-contained application with traditional governance and reusability constraints.
Solution	The large business problem can be broken down into a set of smaller, related problems, allowing the required solution logic to also be decomposed into a corresponding set of smaller, related solution logic units.
Application	Depending on the nature of the large problem, a service-oriented analysis process can be created to cleanly deconstruct it into smaller problems.
Impacts	The ownership of multiple smaller programs can result in increased design complexity and governance challenges.
Principles	n/a
Architecture	Service

Intermediate Routing

By Mark Little, Thomas Rischbeck, Arnaud Simon

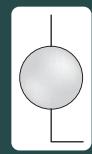


How can dynamic runtime factors affect the path of a message?

Problem	The larger and more complex a service composition is, the more difficult it is to anticipate and design for all possible runtime scenarios in advance, especially with asynchronous, messaging-based communication.
Solution	Message paths can be dynamically determined through the use of intermediary routing logic.
Application	Various types of intermediary routing logic can be incorporated to create message paths based on message content or runtime factors.
Impacts	Dynamically determining a message path adds layers of processing logic and correspondingly can increase performance overhead. Also the use of multiple routing logic can result in overly complex service activities.
Principles	Service Loose Coupling (477), Service Reusability (479), Service Composability (486)
Architecture	Composition

Inventory Endpoint

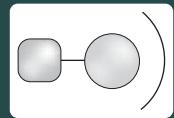
How can a service inventory be shielded from external access while still offering service capabilities to external consumers?



Problem	A group of services delivered for a specific inventory may provide capabilities that are useful to services outside of that inventory. However, for security and governance reasons, it may not be desirable to expose all services or all service capabilities to external consumers.
Solution	Abstract the relevant capabilities into an endpoint service that acts as the official inventory entry point dedicated to a specific set of external consumers.
Application	The endpoint service can expose a contract with the same capabilities as its underlying services, but augmented with policies or other characteristics to accommodate external consumer interaction requirements.
Impacts	Endpoint services can increase the governance freedom of underlying services but can also increase governance effort by introducing redundant service logic and contracts into an inventory.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Abstraction (478)
Architecture	Inventory

Legacy Wrapper

By Thomas Erl, Satadru Roy

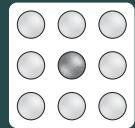


How can wrapper services with non-standard contracts be prevented from spreading indirect consumer-to-implementation coupling?

Problem	Wrapper services required to encapsulate legacy logic are often forced to introduce a non-standard service contract with high technology coupling requirements, resulting in a proliferation of implementation coupling throughout all service consumer programs.
Solution	The non-standard wrapper service can be replaced by or further wrapped with a standardized service contract that extracts, encapsulates, and possibly eliminates legacy technical details from the contract.
Application	A custom service contract and required service logic need to be developed to represent the proprietary legacy interface.
Impacts	The introduction of an additional service adds a layer of processing and associated performance overhead.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Abstraction (478)
Architecture	Service

Logic Centralization

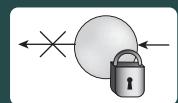
How can the misuse of redundant service logic be avoided?



Problem	If agnostic services are not consistently reused, redundant functionality can be delivered in other services, resulting in problems associated with inventory denormalization and service ownership and governance.
Solution	Access to reusable functionality is limited to official agnostic services.
Application	Agnostic services need to be properly designed and governed, and their use must be enforced via enterprise standards.
Impacts	Organizational issues reminiscent of past reuse projects can raise obstacles to applying this pattern.
Principles	Service Reusability (479), Service Composability (486)
Architecture	Inventory, Composition, Service

Message Screening

By Jason Hogg, Don Smith, Fred Chong, Tom Hollander, Wojtek Kozaczynski, Larry Brader, Nelly Delgado, Dwayne Taylor, Lonnie Wall, Paul Slater, Sajjad Nasir Imran, Pablo Cibraro, Ward Cunningham

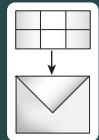


How can a service be protected from malformed or malicious input?

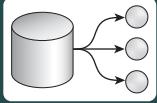
Problem	An attacker can transmit messages with malicious or malformed content to a service, resulting in undesirable behavior.
Solution	The service is equipped or supplemented with special screening routines that assume that all input data is harmful until proven otherwise.
Application	When a service receives a message, it makes a number of checks to screen message content for harmful data.
Impacts	Extra runtime processing is required with each message exchange, and the screening logic requires additional, specialized routines to process binary message content, such as attachments. It may also not be possible to check for all possible forms of harmful content.
Principles	Standardized Service Contract (475)
Architecture	Service

Messaging Metadata

How can services be designed to process activity-specific data at runtime?



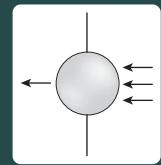
Problem	Because messaging does not rely on a persistent connection between service and consumer, it is challenging for a service to gain access to the state data associated with an overall runtime activity.
Solution	Message contents can be supplemented with activity-specific metadata that can be interpreted and processed separately at runtime.
Application	This pattern requires a messaging framework that supports message headers or properties.
Impacts	The interpretation and processing of messaging metadata adds to runtime performance overhead and increases service activity design complexity.
Principles	Service Loose Coupling (477), Service Statelessness (482)
Architecture	Composition

Metadata Centralization	
<i>How can service metadata be centrally published and governed?</i>	
Problem	Project teams, especially in larger enterprises, run the constant risk of building functionality that already exists or is already in development, resulting in wasted effort, service logic redundancy, and service inventory denormalization.
Solution	Service metadata can be centrally published in a service registry so as to provide a formal means of service registration and discovery.
Application	A private service registry needs to be positioned as a central part of an inventory architecture supported by formal processes for registration and discovery.
Impacts	The service registry product needs to be adequately mature and reliable, and its required use and maintenance needs to be incorporated into all service delivery and governance processes and methodologies.
Principles	Service Discoverability (484)
Architecture	Enterprise, Inventory

Multi-Channel Endpoint

By Satadru Roy

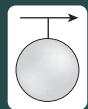
How can legacy logic fragmented and duplicated for different delivery channels be centrally consolidated?



Problem	Legacy systems custom-built for specific delivery channels (mobile phone, desktop, kiosk, etc.) result in redundancy and application silos when multiple channels need to be supported, thereby making these systems burdensome to govern and difficult to federate.
Solution	An intermediary service is designed to encapsulate channel-specific legacy systems and expose a single standardized contract for multiple channel-specific consumers.
Application	The service established by this pattern will require significant processing and workflow logic to support multiple channels while also coordinating interaction with multiple backend legacy systems.
Impacts	The endpoint processing logic established by this pattern often introduces the need for infrastructure upgrades and orchestration-capable middleware and may turn into a performance bottleneck.
Principles	Service Loose Coupling (477), Service Reusability (479)
Architecture	Service

Non-Agnostic Context

How can single-purpose service logic be positioned as an effective enterprise resource?

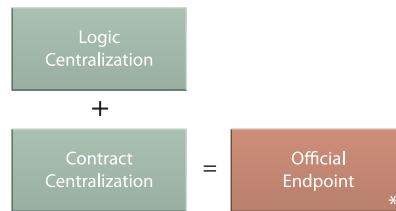


Problem	Non-agnostic logic that is not service-oriented can inhibit the effectiveness of service compositions that utilize agnostic services.
Solution	Non-agnostic solution logic suitable for service encapsulation can be located within services that reside as official members of a service inventory.
Application	A single-purpose functional service context is defined.
Impacts	Although they are not expected to provide reuse potential, non-agnostic services are still subject to the rigor of service-orientation.
Principles	Standardized Service Contract (475), Service Composability (486)
Architecture	Service

Official Endpoint

As important as it is to clearly differentiate Logic Centralization [533] from Contract Centralization [509], it is equally important to understand how these two fundamental patterns can and should be used together.

Applying these two patterns to the same service realizes the Official Endpoint [539] compound pattern. The repeated application of Official Endpoint [539] supports the goal of establishing a federated layer of service endpoints, which is why this compound pattern is also a part of Federated Endpoint Layer [527].

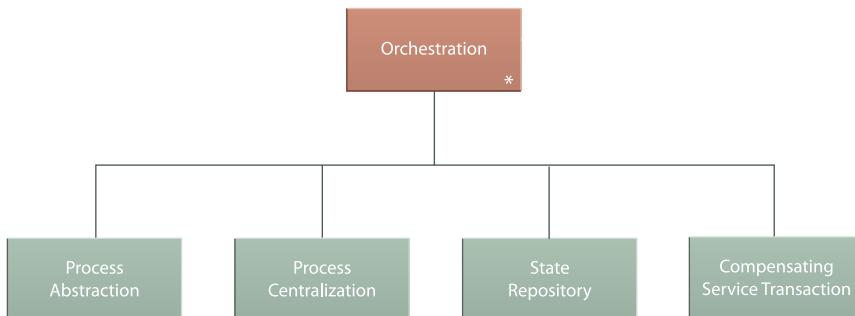


The joint application of Logic Centralization [533] and Contract Centralization [509] results in Official Endpoint.

Orchestration

By Thomas Erl, Brian Loesgen

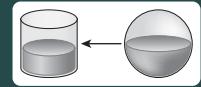
An orchestration platform is dedicated to the effective maintenance and execution of parent business process logic. Modern-day orchestration environments are especially expected to support sophisticated and complex service composition logic that can result in long-running runtime activities.



Orchestration is fundamentally comprised of the co-existent application of Process Abstraction [544], State Repository [567], Process Centralization [545], and Compensating Service Transaction [506], and can be further extended via Atomic Service Transaction [495], Rules Centralization [550], and Data Model Transformation [514].

Partial State Deferral

How can services be designed to optimize resource consumption while still remaining stateful?

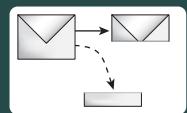


Problem	Service capabilities may be required to store and manage large amounts of state data, resulting in increased memory consumption and reduced scalability.
Solution	Even when services are required to remain stateful, a subset of their state data can be temporarily deferred.
Application	Various state management deferral options exist, depending on the surrounding architecture.
Impacts	Partial state management deferral can add to design complexity and bind a service to the architecture.
Principles	Service Statelessness (482)
Architecture	Inventory, Service

Partial Validation

By David Orchard, Chris Riley

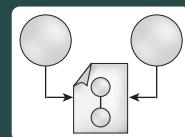
How can unnecessary data validation be avoided?



Problem	The generic capabilities provided by agnostic services sometimes result in service contracts that impose unnecessary data and validation upon consumer programs.
Solution	A consumer program can be designed to only validate the relevant subset of the data and ignore the remainder.
Application	The application of this pattern is specific to the technology used for the consumer implementation. For example, with Web services, XPath can be used to filter out unnecessary data prior to validation.
Impacts	Extra design-time effort is required and the additional runtime data filtering-related logic can reduce the processing gains of avoiding unnecessary validation.
Principles	Standardized Service Contract (475), Service Loose Coupling (477)
Architecture	Composition

Policy Centralization

How can policies be normalized and consistently enforced across multiple services?



Problem	Policies that apply to multiple services can introduce redundancy and inconsistency within service logic and contracts.
Solution	Global or domain-specific policies can be isolated and applied to multiple services.
Application	Up-front analysis effort specific to defining and establishing reusable policies is recommended, and an appropriate policy enforcement framework is required.
Impacts	Policy frameworks can introduce performance overhead and may impose dependencies on proprietary technologies. There is also the risk of conflict between centralized and service-specific policies.
Principles	Standardized Service Contracts (475), Service Loose Coupling (477), Service Abstraction (478)
Architecture	Inventory, Service

Process Abstraction

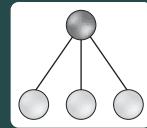
How can non-agnostic process logic be separated and governed independently?



Problem	Grouping task-centric logic together with task-agnostic logic hinders the governance of the task-specific logic and the reuse of the agnostic logic.
Solution	A dedicated parent business process service layer is established to support governance independence and the positioning of task services as potential enterprise resources.
Application	Business process logic is typically filtered out after utility and entity services have been defined, allowing for the definition of task services that comprise this layer.
Impacts	In addition to the modeling and design considerations associated with creating task services, abstracting parent business process logic establishes an inherent dependency on carrying out that logic via the composition of other services.
Principles	Service Loose Coupling (477), Service Abstraction (478), Service Composability (486)
Architecture	Inventory, Composition, Service

Process Centralization

How can abstracted business process logic be centrally governed?



Problem	When business process logic is distributed across independent service implementations, it can be problematic to extend and evolve.
Solution	Logic representing numerous business processes can be deployed and governed from a central location.
Application	Middleware platforms generally provide the necessary orchestration technologies to apply this pattern.
Impacts	Significant infrastructure and architectural changes are imposed when the required middleware is introduced.
Principles	Service Autonomy (481), Service Statelessness (482), Service Composability (486)
Architecture	Inventory, Composition

Protocol Bridging

By Mark Little, Thomas Rischbeck, Arnaud Simon

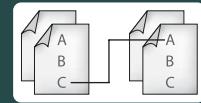


How can a service exchange data with consumers that use different communication protocols?

Problem	Services using different communication protocols or different versions of the same protocol cannot exchange data.
Solution	Bridging logic is introduced to enable communication between different communication protocols by dynamically converting one protocol to another at runtime.
Application	Instead of connecting directly to each other, consumer programs and services connect to a broker, which provides bridging logic that carries out the protocol conversion.
Impacts	Significant performance overhead can be imposed by bridging technologies, and their use can limit or eliminate the ability to incorporate reliability and transaction features.
Principles	Standardized Service Contract (475), Service Composability (486)
Architecture	Inventory, Composition

Proxy Capability

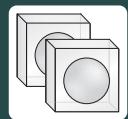
How can a service subject to decomposition continue to support consumers affected by the decomposition?



Problem	If an established service needs to be decomposed into multiple services, its contract and its existing consumers can be impacted.
Solution	The original service contract is preserved, even if underlying capability logic is separated, by turning the established capability definition into a proxy.
Application	Façade logic needs to be introduced to relay requests and responses between the proxy and newly located capabilities.
Impacts	The practical solution provided by this pattern results in a measure of service denormalization.
Principles	Service Loose Coupling (477)
Architecture	Service

Redundant Implementation

How can the reliability and availability of a service be increased?

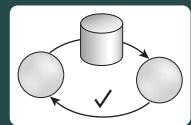


Problem	A service that is being actively reused introduces a potential single point of failure that may jeopardize the reliability of all compositions in which it participates if an unexpected error condition occurs.
Solution	Reusable services can be deployed via redundant implementations or with failover support.
Application	The same service implementation is redundantly deployed or supported by infrastructure with redundancy features.
Impacts	Extra governance effort is required to keep all redundant implementations in synch.
Principles	Service Autonomy (481)
Architecture	Service

Reliable Messaging

By Mark Little, Thomas Rischbeck, Arnaud Simon

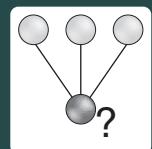
How can services communicate reliably when implemented in an unreliable environment?



Problem	Service communication cannot be guaranteed when using unreliable messaging protocols or when dependent on an otherwise unreliable environment.
Solution	An intermediate reliability mechanism is introduced into the inventory architecture, ensuring that message delivery is guaranteed.
Application	Middleware, service agents, and data stores are deployed to track message deliveries, manage the issuance of acknowledgements, and persist messages during failure conditions.
Impacts	Using a reliability framework adds processing overhead that can affect service activity performance. It also increases composition design complexity and may not be compatible with Atomic Service Transaction [495].
Principles	Service Composability (486)
Architecture	Inventory, Composition

Rules Centralization

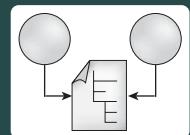
How can business rules be abstracted and centrally governed?



Problem	The same business rules may apply across different business services, leading to redundancy and governance challenges.
Solution	The storage and management of business rules are positioned within a dedicated architectural extension from where they can be centrally accessed and maintained.
Application	The use of a business rules management system or engine is employed and accessed via system agents or a dedicated service.
Impacts	Services are subjected to increased performance overhead, risk, and architectural dependency.
Principles	Service Reusability (479)
Architecture	Inventory

Schema Centralization

How can service contracts be designed to avoid redundant data representation?



Problem	Different service contracts often need to express capabilities that process similar business documents or data sets, resulting in redundant schema content that is difficult to govern.
Solution	Select schemas that exist as physically separate parts of the service contract are shared across multiple contracts.
Application	Up-front analysis effort is required to establish a schema layer independent of and in support of the service layer.
Impacts	Governance of shared schemas becomes increasingly important as multiple services can form dependencies on the same schema definitions.
Principles	Standardized Service Contract (475), Service Loose Coupling (477)
Architecture	Inventory, Service

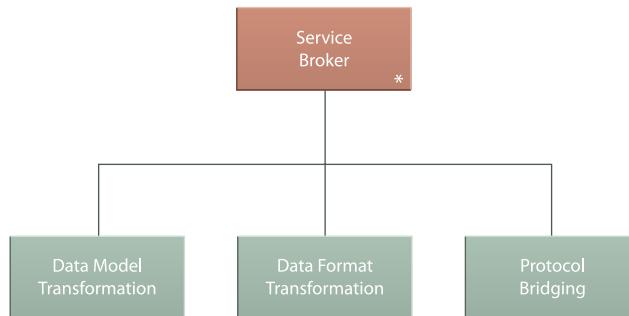
Service Agent	
<i>How can event-driven logic be separated and governed independently?</i>	
Problem	Service compositions can become large and inefficient, especially when required to invoke granular capabilities across multiple services.
Solution	Event-driven logic can be deferred to event-driven programs that don't require explicit invocation, thereby reducing the size and performance strain of service compositions.
Application	Service agents can be designed to automatically respond to predefined conditions without invocation via a published contract.
Impacts	The complexity of composition logic increases when it is distributed across services, and event-driven agents and reliance on service agents can further tie an inventory architecture to proprietary vendor technology.
Principles	Service Loose Coupling (477), Service Reusability (479)
Architecture	Inventory, Composition



Service Broker

By Mark Little, Thomas Rischbeck, Arnaud Simon

Although all of the Service Broker patterns are used only out of necessity, establishing an environment capable of handling the three most common transformation requirements can add a great deal of flexibility to a service-oriented architecture implementation, and also has the added bonus of being able to perform more than one transformation function at the same time.



Service Broker is comprised of the co-existent application of Data Model Transformation [514], Data Format Transformation [513], and Protocol Bridging [546].

Related Patterns in Other Catalogs

Broker (Buschmann, Henney, Schmidt, Meunier, Rohnert, Sommerland, Stal)

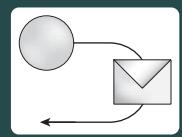
Related Service-Oriented Computing Goals

Increased Intrinsic Interoperability, Increased Vendor Diversification Options, Reduced IT Burden

Service Callback

By Anish Karmarkar

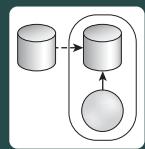
How can a service communicate asynchronously with its consumers?



Problem	When a service needs to respond to a consumer request through the issuance of multiple messages or when service message processing requires a large amount of time, it is often not possible to communicate synchronously.
Solution	A service can require that consumers communicate with it asynchronously and provide a callback address to which the service can send response messages.
Application	A callback address generation and message correlation mechanism needs to be incorporated into the messaging framework and the overall inventory architecture.
Impacts	Asynchronous communication can introduce reliability concerns and can further require that surrounding infrastructure be upgraded to fully support the necessary callback correlation.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Composability (486)
Architecture	Inventory, Service, Composition

Service Data Replication

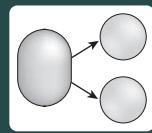
How can service autonomy be preserved when services require access to shared data sources?



Problem	Service logic can be deployed in isolation to increase service autonomy, but services continue to lose autonomy when requiring access to shared data sources.
Solution	Services can have their own dedicated databases with replication to shared data sources.
Application	An additional database needs to be provided for the service and one or more replication channels need to be enabled between it and the shared data sources.
Impacts	This pattern results in additional infrastructure cost and demands, and an excess of replication channels can be difficult to manage.
Principles	Service Autonomy (481)
Architecture	Inventory, Service

Service Decomposition

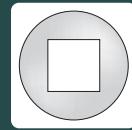
How can the granularity of a service be increased subsequent to its implementation?



Problem	Overly coarse-grained services can inhibit optimal composition design.
Solution	An already implemented coarse-grained service can be decomposed into two or more fine-grained services.
Application	The underlying service logic is restructured, and new service contracts are established. This pattern will likely require Proxy Capability [547] to preserve the integrity of the original coarse-grained service contract.
Impacts	An increase in fine-grained services naturally leads to larger, more complex service composition designs.
Principles	Service Loose Coupling (477), Service Composability (486)
Architecture	Service

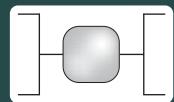
Service Encapsulation

How can solution logic be made available as a resource of the enterprise?



Problem	Solution logic designed for a single application environment is typically limited in its potential to interoperate with or be leveraged by other parts of an enterprise.
Solution	Solution logic can be encapsulated by a service so that it is positioned as an enterprise resource capable of functioning beyond the boundary for which it is initially delivered.
Application	Solution logic suitable for service encapsulation needs to be identified.
Impacts	Service-encapsulated solution logic is subject to additional design and governance considerations.
Principles	n/a
Architecture	Service

Service Façade



How can a service accommodate changes to its contract or implementation while allowing the core service logic to evolve independently?

Problem	The coupling of the core service logic to contracts and implementation resources can inhibit its evolution and negatively impact service consumers.
Solution	A service façade component is used to abstract a part of the service architecture with negative coupling potential.
Application	A separate façade component is incorporated into the service design.
Impacts	The addition of the façade component introduces design effort and performance overhead.
Principles	Standardized Service Contract (475), Service Loose Coupling (477)
Architecture	Service

Service Grid

By David Chappell

How can deferred service state data be scaled and kept fault-tolerant?

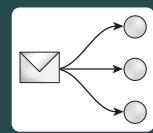


Problem	State data deferred via State Repository or Stateful Services can be subject to performance bottlenecks and failure, especially when exposed to high-usage volumes.
Solution	State data is deferred to a collection of stateful system services that form a grid that provides high scalability and fault tolerance through memory replication and redundancy and supporting infrastructure.
Application	Grid technology is introduced into the enterprise or inventory architecture.
Impacts	This pattern can require a significant infrastructure upgrade and can correspondingly increase governance burden.
Principles	Service Statelessness (482)
Architecture	Enterprise, Inventory, Service

Service Instance Routing

By Anish Karmarkar

How can consumers contact and interact with service instances without the need for proprietary processing logic?



Problem	When required to repeatedly access a specific stateful service instance, consumers must rely on custom logic that more tightly couples them to the service.
Solution	The service provides an instance identifier along with its destination information in a standardized format that shields the consumer from having to resort to custom logic.
Application	The service is still required to provide custom logic to generate and manage instance identifiers, and both service and consumer require a common messaging infrastructure.
Impacts	This pattern can introduce the need for significant infrastructure upgrades and when misused can further lead to overly stateful messaging activities that can violate the Service Statelessness (482) principle.
Principles	Service Loose Coupling (477), Service Statelessness (482), Service Composability (486)
Architecture	Inventory, Composition, Service

Service Layers

How can the services in an inventory be organized based on functional commonality?



Problem	Arbitrarily defining services delivered and governed by different project teams can lead to design inconsistency and inadvertent functional redundancy across a service inventory.
Solution	The inventory is structured into two or more logical service layers, each of which is responsible for abstracting logic based on a common functional type.
Application	Service models are chosen and then form the basis for service layers that establish modeling and design standards.
Impacts	The common costs and impacts associated with design standards and up-front analysis need to be accepted.
Principles	Service Reusability (479), Service Composability (486)
Architecture	Inventory, Service

Service Messaging

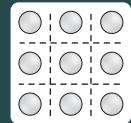
How can services interoperate without forming persistent, tightly coupled connections?



Problem	Services that depend on traditional remote communication protocols impose the need for persistent connections and tightly coupled data exchanges, increasing consumer dependencies and limiting service reuse potential.
Solution	Services can be designed to interact via a messaging-based technology, which removes the need for persistent connections and reduces coupling requirements.
Application	A messaging framework needs to be established, and services need to be designed to use it.
Impacts	Messaging technology brings with it QoS concerns such as reliable delivery, security, performance, and transactions.
Principles	Standardized Service Contract (475), Service Loose Coupling (477)
Architecture	Inventory, Composition, Service

Service Normalization

How can a service inventory avoid redundant service logic?



Problem	When delivering services as part of a service inventory, there is a constant risk that services will be created with overlapping functional boundaries, making it difficult to enable wide-spread reuse.
Solution	The service inventory needs to be designed with an emphasis on service boundary alignment.
Application	Functional service boundaries are modeled as part of a formal analysis process and persist throughout inventory design and governance.
Impacts	Ensuring that service boundaries are and remain well-aligned introduces extra up-front analysis and on-going governance effort.
Principles	Service Autonomy (481)
Architecture	Inventory, Service

Service Perimeter Guard

By Jason Hogg, Don Smith, Fred Chong, Tom Hollander, Wojtek Kozaczynski, Larry Brader, Nelly Delgado, Dwayne Taylor, Lonnie Wall, Paul Slater, Sajjad Nasir Imran, Pablo Cibraro, Ward Cunningham

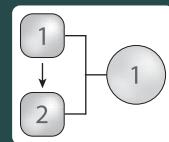


How can services that run in a private network be made available to external consumers without exposing internal resources?

Problem	External consumers that require access to one or more services in a private network can attack the service or use it to gain access to internal resources.
Solution	An intermediate service is established at the perimeter of the private network as a secure contact point for any external consumers that need to interact with internal services.
Application	The service is deployed in a perimeter network and is designed to work with existing firewall technologies so as to establish a secure bridging mechanism between external and internal networks.
Impacts	A perimeter service adds complexity and performance overhead as it establishes an intermediary processing layer for all external-to-internal communication.
Principles	Service Loose Coupling (477), Service Abstraction (478)
Architecture	Service

Service Refactoring

How can a service be evolved without impacting existing consumers?

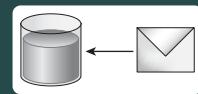


Problem	The logic or implementation technology of a service may become outdated or inadequate over time, but the service has become too entrenched to be replaced.
Solution	The service contract is preserved to maintain existing consumer dependencies, but the underlying service logic and/or implementation are refactored.
Application	Service logic and implementation technology are gradually improved or upgraded but must undergo additional testing.
Impacts	This pattern introduces governance effort as well as risk associated with potentially negative side-effects introduced by new logic or technology.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Abstraction (478)
Architecture	Service

State Messaging

By Anish Karmarkar

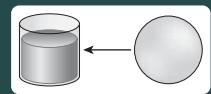
How can a service remain stateless while participating in stateful interactions?



Problem	When services are required to maintain state information in memory between message exchanges with consumers, their scalability can be comprised, and they can become a performance burden on the surrounding infrastructure.
Solution	Instead of retaining the state data in memory, its storage is temporarily delegated to messages.
Application	Depending on how this pattern is applied, both services and consumers may need to be designed to process message-based state data.
Impacts	This pattern may not be suitable for all forms of state data, and should messages be lost, any state information they carried may be lost as well.
Principles	Standardized Service Contract (475), Service Statelessness (482), Service Composability (486)
Architecture	Composition, Service

State Repository

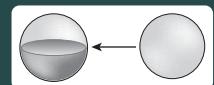
How can service state data be persisted for extended periods without consuming service runtime resources?



Problem	Large amounts of state data cached to support the activity within a running service composition can consume too much memory, especially for long-running activities, thereby decreasing scalability.
Solution	State data can be temporarily written to and then later retrieved from a dedicated state repository.
Application	A shared or dedicated repository is made available as part of the inventory or service architecture.
Impacts	The addition of required write and read functionality increases the service design complexity and can negatively affect performance.
Principles	Service Statelessness (482)
Architecture	Inventory, Service

Stateful Services

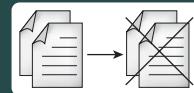
How can service state data be persisted and managed without consuming service runtime resources?



Problem	State data associated with a particular service activity can impose a great deal of runtime state management responsibility upon service compositions, thereby reducing their scalability.
Solution	State data is managed and stored by intentionally stateful utility services.
Application	Stateful utility services provide in-memory state data storage and/or can maintain service activity context data.
Impacts	If not properly implemented, stateful utility services can become a performance bottleneck.
Principles	Service Statelessness (482)
Architecture	Inventory, Service

Termination Notification

By David Orchard, Chris Riley

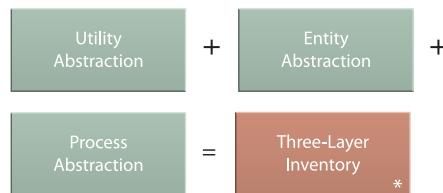


How can the scheduled expiry of a service contract be communicated to consumer programs?

Problem	Consumer programs may be unaware of when a service or a service contract version is scheduled for retirement, thereby risking runtime failure.
Solution	Service contracts can be designed to express termination information for programmatic and human consumption.
Application	Service contracts can be extended with ignorable policy assertions or supplemented with human-readable annotations.
Impacts	The syntax and conventions used to express termination information must be understood by service consumers in order for this information to be effectively used.
Principles	Standardized Service Contract (475)
Architecture	Composition, Service

Three-Layer Inventory

This compound pattern is simply comprised of the combined application of the three service layer patterns. Three-Layer Inventory exists because the combined application of these three patterns results in common layers of abstraction that have been proven to complement and support each other by establishing services with flexible variations of agnostic and non-agnostic functional contexts.



The joint application of Utility Abstraction [573], Entity Abstraction [524], and Process Abstraction [544] results in Three-Layer Inventory.

Trusted Subsystem

By Jason Hogg, Don Smith, Fred Chong, Tom Hollander, Wojtek Kozaczynski, Larry Brader, Nelly Delgado, Dwayne Taylor, Lonnie Wall, Paul Slater, Sajjad Nasir Imran, Pablo Cibraro, Ward Cunningham

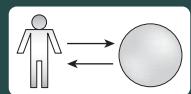


How can a consumer be prevented from circumventing a service and directly accessing its resources?

Problem	A consumer that accesses backend resources of a service directly can compromise the integrity of the resources and can further lead to undesirable forms of implementation coupling.
Solution	The service is designed to use its own credentials for authentication and authorization with backend resources on behalf of consumers.
Application	Depending on the nature of the underlying resources, various design options and security technologies can be applied.
Impacts	If this type of service is compromised by attackers or unauthorized consumers, it can be exploited to gain access to a wide range of downstream resources.
Principles	Service Loose Coupling (477)
Architecture	Service

UI Mediator

By Clemens Utschig-Utschig, Berthold Maier,
Bernd Trops, Hajo Normann, Torsten Winterberg



How can a service-oriented solution provide a consistent, interactive user experience?

Problem	Because the behavior of individual services can vary depending on their design, runtime usage, and the workload required to carry out a given capability, the consistency with which a service-oriented solution can respond to requests originating from a user-interface can fluctuate, leading to a poor user experience.
Solution	Establish mediator logic solely responsible for ensuring timely interaction and feedback with user-interfaces and presentation logic.
Application	A utility mediator service or service agent is positioned as the initial recipient of messages originating from the user-interface. This mediation logic responds in a timely and consistent manner regardless of the behavior of the underling solution.
Impacts	The mediator logic establishes an additional layer of processing that can add to the required runtime processing.
Principles	Service Loose Coupling (477)
Architecture	Composition

Utility Abstraction

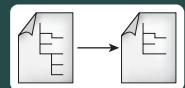
How can common non-business centric logic be separated, reused, and independently governed?



Problem	When non-business centric processing logic is packaged together with business-specific logic, it results in the redundant implementation of common utility functions across different services.
Solution	A service layer dedicated to utility processing is established, providing reusable utility services for use by other services in the inventory.
Application	The utility service model is incorporated into analysis and design processes in support of utility logic abstraction, and further steps are taken to define balanced service contexts.
Impacts	When utility logic is distributed across multiple services it can increase the size, complexity, and performance demands of compositions.
Principles	Service Loose Coupling (477), Service Abstraction (478), Service Reusability (479), Service Composability (486)
Architecture	Inventory, Composition, Service

Validation Abstraction

How can service contracts be designed to more easily adapt to validation logic changes?

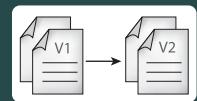


Problem	Service contracts that contain detailed validation constraints become more easily invalidated when the rules behind those constraints change.
Solution	Granular validation logic and rules can be abstracted away from the service contract, thereby decreasing constraint granularity and increasing the contract's potential longevity.
Application	Abstracted validation logic and rules need to be moved to the underlying service logic, a different service, a service agent, or elsewhere.
Impacts	This pattern can somewhat decentralize validation logic and can also complicate schema standardization.
Principles	Standardized Service Contract (475), Service Loose Coupling (477), Service Abstraction (478)
Architecture	Service

Version Identification

By David Orchard, Chris Riley

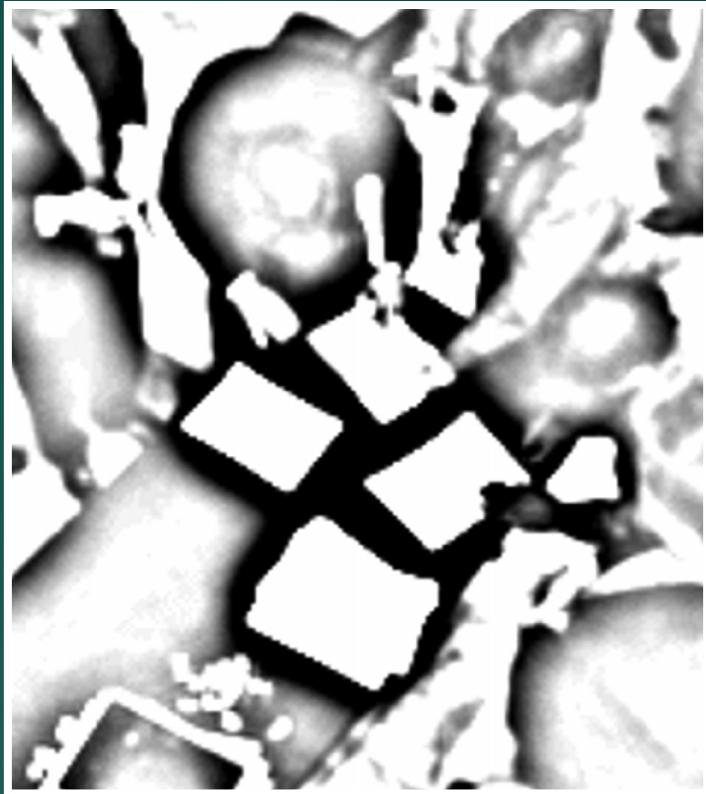
How can consumers be made aware of service contract version information?



Problem	When an already-published service contract is changed, unaware consumers will miss the opportunity to leverage the change or may be negatively impacted by the change.
Solution	Versioning information pertaining to compatible and incompatible changes can be expressed as part of the service contract, both for communication and enforcement purposes.
Application	With Web service contracts, version numbers can be incorporated into namespace values and as annotations.
Impacts	This pattern may require that version information be expressed with a proprietary vocabulary that needs to be understood by consumer designers in advance.
Principles	Standardized Service Contract (475)
Architecture	Service

This page intentionally left blank

Appendix E



The Annotated SOA Manifesto

The SOA Manifesto was authored and announced during the 2nd Annual International SOA Symposium in Rotterdam by a working group comprised of 17 experts and thought leaders from different organizations. Two of the SOA Manifesto Working Group members (Anne Thomas Manes and Thomas Erl) are co-authors of this book.

The original SOA Manifesto is published at www.soa-manifesto.org. You are encouraged to visit this site and enter your name on the *Become a Signatory* form to show your support for the values and principles declared in the manifesto.

Subsequent to the announcement of the SOA Manifesto, Thomas Erl authored an annotated version that supplements individual statements from the original manifesto with additional commentary and insights. The Annotated SOA Manifesto is published at www.soa-manifesto.com and has been further provided as a supplementary resource in this appendix.

The Annotated SOA Manifesto

Commentary and Insights about the SOA Manifesto from Thomas Erl

Service-orientation is a paradigm that frames what you do. Service-oriented architecture (SOA) is a type of architecture that results from applying service-orientation.

From the beginning it was understood that this was to be a manifesto about two distinct yet closely related topics: the service-oriented architectural model and service-orientation, the paradigm through which the architecture is defined. The format of this manifesto was modeled after the Agile Manifesto, which limits content to concise statements that express ambitions, values, and guiding principles for realizing those ambitions and values. Such a manifesto is not a specification, a reference model or even a white paper, and without an option to provide actual definitions, we decided to add this preamble in order to clarify how and why these terms are referenced in other parts of the manifesto document.

We have been applying service-orientation...

The service-orientation paradigm is best viewed as a method or an approach for realizing a specific target state that is further defined by a set of strategic goals and benefits. When we apply service-orientation, we shape software programs and technology architecture in support of realizing this target state. This is what qualifies technology architecture as being service-oriented.

...to help organizations consistently deliver sustainable business value, with increased agility and cost effectiveness...

This continuation of the preamble highlights some of the most prominent and commonly expected strategic benefits of service-oriented computing. Understanding these benefits helps shed some light on the aforementioned target state we intend to realize as a result of applying service-orientation.

Agility at a business level is comparable to an organization's responsiveness. The more easily and effectively an organization can respond to business change, the more efficient and successful it will be to adapting to the impacts of the change (and further leverage whatever benefits the change may bring about).

Service-orientation positions services as IT assets that are expected to provide repeated value over time that far exceeds the initial investment required for their delivery. Cost-effectiveness relates primarily to this expected return on investment. In many ways, an increase in cost-effectiveness goes hand-in-hand with an increase in agility; if there is more opportunity to reuse existing services, then there is generally less expense required to build new solutions.

“Sustainable” business value refers to the long-term goals of service-orientation to establish software programs as services with the inherent flexibility to be continually composed into new solution configurations and evolved to accommodate ever-changing business requirements.

...in line with changing business needs.

These last six words of the preamble are key to understanding the underlying philosophy of service-oriented computing. The need to accommodate business change on an on-going basis is foundational to service-orientation and considered a fundamental over-arching strategic goal.

Through our work we have come to prioritize:

The upcoming statements establish a core set of values, each of which is expressed as a prioritization over something that is also considered of value. The intent of this value system is to address the hard choices that need to be made on a regular basis in order for the strategic goals and benefits of service-oriented computing to be consistently realized.

Business value over technical strategy

As stated previously, the need to accommodate business change is an overarching strategic goal. Therefore, the foundational quality of service-oriented architecture and of any software programs, solutions, and eco-systems that result from the adoption of service-orientation is that they are business-driven. It is not about technology determining the direction of the business, it is about the business vision dictating the utilization of technology.

This priority can have a profound ripple effect within the regions of an IT enterprise. It introduces changes to just about all parts of IT delivery lifecycles, from how we plan for and fund automation solutions, to how we build and govern them. All other values and principles in the manifesto, in one way or another, support the realization of this value.

Strategic goals over project-specific benefits

Historically, many IT projects focused solely on building applications designed specifically to automate business process requirements that were current at that time. This fulfilled immediate (tactical) needs, but as more of these single-purpose applications were delivered, it resulted in an IT enterprise filled with islands of logic and data referred to as application “silos.” As new business requirements would emerge, either new silos were created or integration channels between silos were established. As yet more business change arose, integration channels had to be augmented, even more silos had to be created, and soon the IT enterprise landscape became convoluted and increasingly burdensome, expensive, and slow to evolve.

In many ways, service-orientation emerged in response to these problems. It is a paradigm that provides an alternative to project-specific, silo-based, and integrated application development by adamantly prioritizing the attainment of long-term, strategic business goals. The target state advocated by service-orientation does not have traditional application silos. And even when legacy resources and application silos exist in environments where service-orientation is adopted, the target state is one where they are harmonized to whatever extent feasible.

Intrinsic interoperability over custom integration

For software programs to share data they need to be interoperable. If software programs are not designed to be compatible, they will likely not be interoperable. To enable interoperability between incompatible software programs requires that they be integrated. Integration is therefore the effort required to achieve interoperability between disparate software programs.

Although often necessary, customized integration can be expensive and time consuming and can lead to fragile architectures that are burdensome to evolve. One of the goals of service-orientation is to minimize the need for customized integration by shaping software programs (within a given domain) so that they are natively compatible. This is a quality referred to as intrinsic interoperability. The service-orientation paradigm encompasses a set of specific design principles that are geared toward establishing intrinsic interoperability on several levels.

Intrinsic interoperability, as a characteristic of software programs that reside within a given domain, is key to realizing strategic benefits, such as increased cost-effectiveness and agility.

Shared services over specific-purpose implementations

As just explained, service-orientation establishes a design approach comprised of a set of design principles. When applied to a meaningful extent, these principles shape a software program into a unit of service-oriented logic that can be legitimately referred to as a service.

Services are equipped with concrete characteristics (such as those that enable intrinsic interoperability) that directly support the previously described target state. One of these characteristics is the encapsulation of multi-purpose logic that can be shared and reused in support of the automation of different business processes.

A shared service establishes itself as an IT asset that can provide repeated business value while decreasing the expense and effort to deliver new automation solutions. While there is value in traditional, single-purpose applications that solve tactical business requirements, the use of shared services provides greater value in realizing strategic goals of service-oriented computing (which again include an increase in cost-effectiveness and agility).

Flexibility over optimization

This is perhaps the broadest of the value prioritization statements and is best viewed as a guiding philosophy for how to better prioritize various considerations when delivering and evolving individual services and inventories of services.

Optimization primarily refers to the fulfillment of tactical gains by tuning a given application design or expediting its delivery to meet immediate needs. There is nothing undesirable about this, except that it can lead to the aforementioned silo-based environments when not properly prioritized in relation to fostering flexibility.

For example, the characteristic of flexibility goes beyond the ability for services to effectively (and intrinsically) share data. To be truly responsive to ever-changing business requirements, services must also be flexible in how they can be combined and aggregated into composite solutions. Unlike traditional distributed applications that often were relatively static despite the fact that they were componentized, service compositions need to be designed with a level of inherent flexibility that allows for constant augmentation. This means that when an existing business process changes or when a new business process is introduced, we need to be able to add, remove, and extend services within the composition architecture with minimal (integration) effort. This is why service composability is one of the key service-orientation design principles.

Evolutionary refinement over pursuit of initial perfection

There is a common point of confusion when it comes to the term “agility” in relation to service-orientation. Some design approaches advocate the rapid delivery of software programs for immediate gains. This can be considered “tactical agility,” as the focus is on tactical, short-term benefit. Service-orientation advocates the attainment of agility on an organizational or business level with the intention of empowering the organization, as a whole, to be responsive to change. This form of organizational agility can also be referred to as “strategic agility” because the emphasis is on longevity in that, with every software program we deliver, we want to work toward a target state that fosters agility with long-term strategic value.

For an IT enterprise to enable organizational agility, it must evolve in tandem with the business. We generally cannot predict how a business will need to evolve over time and therefore we cannot initially build the perfect services. At the same time, there is usually a wealth of knowledge that already exists within an organization’s existing business intelligence that can be harvested during the analysis and modeling stages of SOA projects.

This information, together with service-orientation principles and proven methodologies, can help us identify and define a set of services that capture how the business exists and operates today while being sufficiently flexible to adapt to how the business changes over time.

That is, while we value the items on the right, we value the items on the left more.

By studying how these values are prioritized, we gain insight into what distinguishes service-orientation from other paradigms. This type of insight can benefit IT practitioners in several ways. For example, it can help establish fundamental criteria that we can use to determine how compatible service-orientation is for a given organization or IT enterprise. It can further help determine the extent to which service-orientation can or should be adopted.

An appreciation of the core values can also help us understand how challenging it may be to successfully carry out SOA projects within certain environments. For example, several of these prioritizations may clash head-on with established beliefs and preferences. In such a case, the benefits of service-orientation need to be weighed against the effort and impact their adoption may have (not just on technology, but also on the organization and IT culture).

The upcoming guiding principles were provided to help address many of these types of challenges.

We follow these principles:

So far, the manifesto has established an overall vision as well as a set of core values associated with the vision. The remainder of the declaration is comprised of a set of principles that are provided as guidance for adhering to the values and realizing the vision.

It's important to keep in mind that these are guiding principles specific to this manifesto. There is a separate set of established design principles that comprise the service-orientation design paradigm and there are many more documented practices and patterns specific to service-orientation and service-oriented architecture.

Respect the social and power structure of the organization.

One of the most common SOA pitfalls is approaching adoption as a technology-centric initiative. Doing so almost always leads to failure because we are simply not prepared for the inevitable organizational impacts.

The adoption of service-orientation is about transforming the way we automate business. However, regardless of what plans we may have for making this transformation

effort happen, we must always begin with an understanding and an appreciation of the organization, its structure, its goals, and its culture.

The adoption of service-orientation is very much a human experience. It requires support from those in authority and then asks that an IT culture adopt a strategic, community-centric mindset. We must fully acknowledge and plan for this level of organizational change in order to receive the necessary long-term commitments required to achieve the target state of service-orientation.

These types of considerations not only help us determine how to best proceed with an SOA initiative, they further assist us in defining the most appropriate scope and approach for adoption.

Recognize that SOA ultimately demands change on many levels.

There's a saying that goes: "Success is being prepared for opportunity." Perhaps the number one lesson learned from SOA projects carried out so far is that we must fully comprehend and then plan and prepare for the volume and range of change that is brought about as a result of adopting service-orientation. Here are some examples.

Service-orientation changes how we build automation solutions by positioning software programs as IT assets with long-term, repeatable business value. An upfront investment is required to create an environment comprised of such assets and an on-going commitment is required to maintain and leverage their value. So, right out of the gate, changes are required to how we fund, measure, and maintain systems within the IT enterprise.

Furthermore, because service-orientation introduces services that are positioned as resources of the enterprise, there will be changes in how we own different parts of systems and regulate their design and usage, not to mention changes to the infrastructure required to guarantee continuous scalability and reliability.

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

A common myth has been that in order to realize the strategic goals of service-oriented computing, service-orientation must be adopted on an enterprise-wide basis. This means establishing and enforcing design and industry standards across the IT enterprise so as to create an enterprise-wide inventory of intrinsically interoperable services. While there is nothing wrong with this ideal, it is not a realistic goal for many organizations, especially those with larger IT enterprises.

The most appropriate scope for any given SOA adoption effort needs to be determined as a result of planning and analysis in conjunction with pragmatic considerations, such as the aforementioned impacts on organizational structures, areas of authority, and cultural changes that are brought about.

These types of factors help us determine a scope of adoption that is manageable. But for any adoption effort to result in an environment that progresses the IT enterprise toward the desired strategic target state, the scope must also be meaningful. In other words, it must be meaningfully cross-silo so that collections of services can be delivered in relation to each other within a pre-defined boundary. In other words, we want to create “continents of services,” not the dreaded “islands of services.”

This concept of building independently owned and governed service inventories within domains of the same IT enterprise reduces many of the risks that are commonly attributed to “big-bang” SOA projects and furthermore mitigates the impact of both organizational and technological changes (because the impact is limited to a segmented and managed scope). It is also an approach that allows for phased adoption where one domain service inventory can be established at a time.

Products and standards alone will neither give you SOA nor apply the service-orientation paradigm for you.

This principle addresses two separate but very much related myths. The first is that you can buy your way into SOA with modern technology products, and the second is the assumption that the adoption of industry standards (such as XML, WSDL, SCA, etc.) will naturally result in service-oriented technology architecture.

The vendor and industry standards communities have been credited with building modern service technology innovation upon non-proprietary frameworks and platforms. Everything from service virtualization to cloud computing and grid computing has helped advance the potential for building sophisticated and complex service-oriented solutions. However, none of these technologies are exclusive to SOA. You can just as easily build silo-based systems in the cloud as you can on your own private servers.

There is no such thing as “SOA in a box” because in order to achieve service-oriented technology architecture, service-orientation needs to be successfully applied; this, in turn, requires that everything we design and build be driven by the unique direction, vision, and requirements of the business.

SOA can be realized through a variety of technologies and standards.

Service-orientation is a technology-neutral and vendor-neutral paradigm. Service-oriented architecture is a technology-neutral and vendor neutral architectural model. Service-oriented computing can be viewed as a specialized form of distributed computing. Service-oriented solutions can therefore be built using just about any technologies and industry standards suitable for distributed computing.

While some technologies (especially those based on industry standards) can increase the potential of applying some service-orientation design principles, it is really the potential to fulfill business requirements that ultimately determines the most suitable choice of technologies and industry standards.

Establish a uniform set of enterprise standards and policies based on industry, de facto, and community standards.

Industry standards represent non-proprietary technology specifications that help establish, among other things, consistent baseline characteristics (such as transport, interface, message format, etc.) of technology architecture. However, the use of industry standards alone does not guarantee that services will be intrinsically interoperable.

For two software programs to be fully compatible, additional conventions (such as data models and policies) need to be adhered to. This is why IT enterprises must establish and enforce design standards. Failure to properly standardize and regulate the standardization of services within a given domain will begin to tear at the fabric of interoperability upon which the realization of many strategic benefits relies.

This principle not only advocates the use of enterprise design standards, it also reminds us that, whenever possible and feasible, custom design standards should be based upon and incorporate standards already in use by the industry and the community in general.

Pursue uniformity on the outside while allowing diversity on the inside.

Federation can be defined as the unification of a set of disparate entities. While allowing each entity to be independently governed on the inside, all agree to adhere to a common, unified front.

A fundamental part of service-oriented architecture is the introduction of a federated endpoint layer that abstracts service implementation details while publishing a set of endpoints that represent individual services within a given domain in a unified manner. Accomplishing this generally involves achieving unity based on a combination of industry and design standards. The consistency of this unity across services is key to realizing intrinsic interoperability.

A federated endpoint layer further helps increase opportunities to explore vendor-diversity options. For example, one service may need to be built upon a completely different platform than another. As long as these services maintain compatible endpoints, the governance of their respective implementations can remain independent. This not only highlights that services can be built using different implementation mediums (such as EJB, .NET, SOAP, REST, etc.), it also emphasizes that different intermediary platforms and technologies can be utilized together, as required.

Note that this type of diversity comes with a price. This principle does not advocate diversification itself—it simply recommends that we allow diversification when justified, so that “best-of-breed” technologies and platforms can be leveraged to maximize business requirements fulfillment.

Identify services through collaboration with business and technology stakeholders.

In order for technology solutions to be business-driven, the technology must be in synch with the business. Therefore, another goal of service-oriented computing is to align technology and business. The stage at which this alignment is initially accomplished is during the analysis and modeling processes that usually precede actual service development and delivery.

The critical ingredient to carrying out service-oriented analysis is to have both business and technology experts working hand-in-hand to identify and define candidate services. For example, business experts can help accurately define functional contexts pertaining to business-centric services, while technology experts can provide pragmatic input to ensure that the granularity and definition of conceptual services remains realistic in relation to their eventual implementation environments.

Maximize service usage by considering the current and future scope of utilization.

The extent of a given SOA project may be enterprise-wide or it may be limited to a domain of the enterprise. Whatever the scope, a pre-defined boundary is established to encompass an inventory of services that need to be conceptually modeled before they can be developed. By modeling multiple services in relation to each other we essentially establish a blueprint of the services we will eventually be building. This modeling exercise is critical when attempting to identify and define services that can be shared by different solutions.

There are various methodologies and approaches that can be used to carry out service-oriented analysis stages. However, a common thread among all of them is that the functional boundaries of services be normalized to avoid redundancy. Even then,

normalized services do not necessarily make for highly reusable services. Other factors come into play, such as service granularity, autonomy, state management, scalability, compositability, and the extent to which service logic is sufficiently generic so that it can be effectively reused.

These types of considerations guided by business and technology expertise provide the opportunity to define services that capture current utilization requirements while having the flexibility to adapt to future change.

Verify that services satisfy business requirements and goals.

As with anything, services can be misused. When growing and managing a portfolio of services, their usage and effectiveness at fulfilling business requirements need to be verified and measured. Modern tools provide various means of monitoring service usage, but there are intangibles that also need to be taken into consideration to ensure that services are not just used because they are available, but to verify that they are truly fulfilling business needs and meeting expectations.

This is especially true with shared services that shoulder multiple dependencies. Not only do shared services require adequate infrastructure to guarantee scalability and reliability for all of the solutions that reuse them, they also need to be designed and extended with great care to ensure their functional contexts are never skewed.

Evolve services and their organization in response to real use.

This guiding principle ties directly back to the “Evolutionary refinement over pursuit of initial perfection” value statement, as well as the overall goal of maintaining an alignment of business and technology.

We can never expect to rely on guesswork when it comes to determining service granularity, the range of functions that services need to perform, or how services will need to be organized into compositions. Based on whatever extent of analysis we are able to initially perform, a given service will be assigned a defined functional context and will contain one or more functional capabilities that likely involve it in one or more service compositions.

As real world business requirements and circumstances change, the service may need to be augmented, extended, refactored, or perhaps even replaced. Service-orientation design principles build native flexibility into service architectures so that, as software programs, services are resilient and adaptive to change and to being changed in response to real world usage.

Separate the different aspects of a system that change at different rates.

What makes monolithic and silo-based systems inflexible is that change can have a significant impact on their existing usage. This is why it is often easier to create new silo-based applications rather than augment or extend existing ones.

The rationale behind the separation of concerns (a commonly known software engineering theory) is that a larger problem can be more effectively solved when decomposed into a set of smaller problems or concerns. When applying service-orientation to the separation of concerns, we build corresponding units of solution logic that solve individual concerns, thereby allowing us to aggregate the units to solve the larger problem in addition to giving us the opportunity to aggregate them into different configurations in order to solve other problems.

Besides fostering service reusability, this approach introduces numerous layers of abstraction that help shield service-comprised systems from the impacts of change. This form of abstraction can exist at different levels. For example, if legacy resources encapsulated by one service need to be replaced, the impact of that change can be mitigated as long as the service is able to retain its original endpoint and functional behavior.

Another example is the separation of agnostic from non-agnostic logic. The former type of logic has high reuse potential if it is multi-purpose and less likely to change. Non-agnostic logic, on the other hand, typically represents the single-purpose parts of parent business process logic, which are often more volatile. Separating these respective logic types into different service layers further introduces abstraction that enables service reusability while shielding services, and any solutions that utilize them, from the impacts of change.

Reduce implicit dependencies and publish all external dependencies to increase robustness and reduce the impact of change.

One of the most well-known service-orientation design principles is that of service loose coupling. How a service architecture is internally structured and how services relate to programs that consume them (which can include other services) all comes down to dependencies that are formed on individually moving parts that are part of the service architecture.

Layers of abstraction help ease evolutionary change by localizing the impacts of the change to controlled regions. For example, within service architectures, service facades can be used to abstract parts of the implementation in order to minimize the reach of implementation dependencies.

On the other hand, published technical service contracts need to disclose the dependencies that service consumers must form in order to interact with services. By reducing internal dependencies that can affect these technical contracts when change does occur, we avoid proliferating the impact of those changes upon dependent service consumers.

At every level of abstraction, organize each service around a cohesive and manageable unit of functionality.

Each service requires a well-defined functional context that determines what logic does and does not belong within the service's functional boundary. Determining the scope and granularity of these functional service boundaries is one of the most critical responsibilities during the service delivery lifecycle.

Services with coarse functional granularity may be too inflexible to be effective, especially if they are expected to be reusable. On the other hand, overly fine grained services may tax an infrastructure in that service compositions will need to consist of increased quantities of composition members.

Determining the right balance of functional scope and granularity requires a combination of business and technology expertise, and further requires an understanding of how services within a given boundary relate to each other.

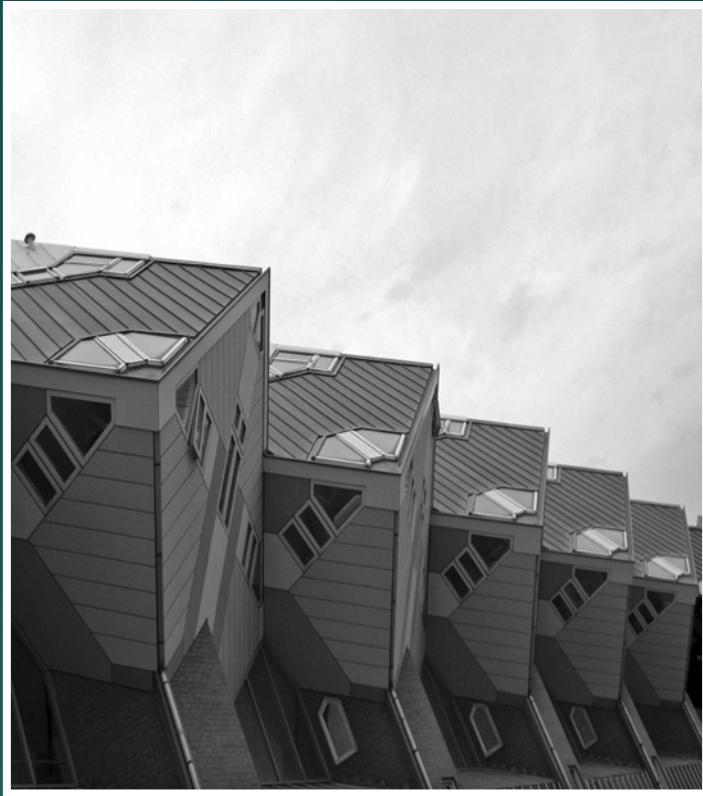
Many of the guiding principles described in this manifesto will help in making this determination in support of positioning each service as an IT asset capable of furthering an IT enterprise toward that target state whereby the strategic benefits of service-oriented computing are realized.

Ultimately, though, it will always be the attainment of real world business value that dictates, from conception to delivery to repeated usage, the evolutionary path of any unit of service-oriented functionality.

—*Thomas Erl (November 22, 2009)*

www.soa-manifesto.com

Appendix F



Versioning Fundamentals for Web Services and REST Services

F.1 Versioning Basics

F.2 Versioning and Compatibility

F.3 REST Service Compatibility Considerations

F.4 Version Identifiers

F.5 Versioning Strategies

F.6 REST Service Versioning Considerations

NOTE

The following appendix contains the original Chapter 21 from the book *Web Service Contract Design and Versioning for SOA*, further enhanced with new content specific to REST service versioning. This content has been provided as a supplementary resource to elaborate upon some of the governance topics pertaining to the Service Versioning and Retirement project stage.

The upcoming sections contain various references to chapters in the *Web Service Contract Design and Versioning for SOA* book. These chapter references are always qualified with that book title, so be sure to disregard these references in relation to this book.

Also, this appendix contains a number of examples with markup code. You are not required to understand these examples, nor are you required to learn the referenced markup languages in order to use any other part of this book. The code examples are provided solely to preserve the structure of the original chapter from the *Web Service Contract Design and Versioning for SOA* book, which is a title authored primarily for Service Architects, Service Developers, Schema Custodians, and Policy Custodians.

After a service contract is deployed, consumer programs will naturally begin forming dependencies on it. When we are subsequently forced to make changes to the contract, we need to figure out:

- whether the changes will negatively impact existing (and potentially future) service consumers
- how changes that will and will not impact consumers should be implemented and communicated

These issues result in the need for versioning. Any time you introduce the concept of versioning into an SOA project, a number of questions will likely be raised, for example:

- What exactly constitutes a new version of a service contract? What's the difference between a major and minor version?

- What do the parts of a version number indicate?
- Will the new version of the contract still work with existing consumers that were designed for the old contract version?
- Will the current version of the contract work with new consumers that may have different data exchange requirements?
- What is the best way to add changes to existing contracts while minimizing the impact on consumers?
- Will we need to host old and new contracts at the same time? If yes, for how long?

We will address these questions and provide a set of options for solving common versioning problems. The upcoming sections begin by covering some basic concepts, terminology, and strategies specific to service contract versioning.

F.1 Versioning Basics

So when we say that we're creating a new version of a service contract, what exactly are we referring to? The following sections explain some fundamental terms and concepts and further distinguish between Web service contracts and REST service contracts.

Versioning Web Services

As we've established many times over in this book, a Web service contract can be comprised of several individual documents and definitions that are linked and assembled together to form a complete technical interface.

For example, a given Web service contract can consist of:

- one (sometimes more) WSDL definitions
- one (usually more) XML Schema definitions
- some (sometimes no) WS-Policy definitions

Furthermore, each of these definition documents can be shared by other Web service contracts.

For example:

- a centralized XML Schema definition will commonly be used by multiple WSDL definitions

- a centralized WS-Policy definition will commonly be applied to multiple WSDL definitions
- an abstract WSDL description can be imported by multiple concrete WSDL descriptions or vice versa

Of all the different parts of a Web service contract, the part that establishes the fundamental technical interface is the abstract description of the WSDL definition. This represents the core of a Web service contract and is then further extended and detailed through schema definitions, policy definitions, and one or more concrete WSDL descriptions.

When we need to create a new version of a Web service contract, we can therefore assume that there has been a change in the abstract WSDL description or one of the contract documents that relates to the abstract WSDL description. How the different constructs of a WSDL can be versioned is covered in Chapter 21 (*Web Service Contract Design and Versioning for SOA*).

The Web service contract content commonly subject to change is the XML schema content that provides the types for the abstract description's message definitions. Chapter 22 (*Web Service Contract Design and Versioning for SOA*) explores the manner in which the underlying schema definitions for messages can be changed and evolved.

Finally, the one other contract-related technology that can still impose versioning requirements but is less likely to do so simply because it is a less common part of Web service contracts is WS-Policy. How policies in general relate to contract versioning is explained as part of the advanced topics in Chapter 23 (*Web Service Contract Design and Versioning for SOA*).

Versioning REST Services

If we follow the REST model of using a uniform contract to express service capabilities, the sharing of definition documents between service contract is even clearer.

For example:

- all HTTP methods used in contracts are standard across the architecture
- XML Schema definitions are standard, as they are wrapped up in general media types
- the identifier syntax for lightweight service endpoints (known as resources) are standard across the architecture

Changes to the uniform contract facets that underlie each service contract can impact any REST service in the service inventory.

Fine and Coarse-Grained Constraints

Regardless of whether XML schemas are used with Web services or REST services, versioning changes are often tied to the increase or reduction of the quantity or granularity of constraints expressed in the schema definition. Therefore, let's briefly recap the meaning of the term constraint granularity in relation to a type definition.

Note the bolded and italicized parts in the following example:

```
<xsd:element name="LineItem" type="LineItemType"/>
<xsd:complexType name="LineItemType">
    <xsd:sequence>
        <xsd:element name="productID" type="xsd:string"/>
        <xsd:element name="productName" type="xsd:string"/>
        <xsd:any minOccurs="0" maxOccurs="unbounded"
            namespace="#any" processContents="lax"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="#any"/>
</xsd:complexType>
```

Example F.1

A complexType construct containing fine and coarse-grained constraints.

As indicated by the bolded text, there are elements with specific names and data types that represent parts of the message definition with a *fine* level of constraint granularity. All of the message instances (the actual XML documents that will be created based on this structure) must conform to these constraints in order to be considered valid (which is why these are considered the absolute “minimum” constraints).

The italicized text shows the element and attribute wildcards also contained by this complex type. These represent parts of the message definition with an extremely *coarse* level of constraint granularity in that messages do not need to comply to these parts of the message definition at all.

The use of the terms “fine-grained” and “coarse-grained” is highly subjective. What may be a fine-grained constraint in one contract may not be in another. The point is to understand how these terms can be applied when comparing parts of a message definition or when comparing different message definitions with each other.

F.2 Versioning and Compatibility

The number one concern when developing and deploying a new version of a service contract is the impact it will have on other parts of the enterprise that have formed or will form dependencies on it. This measure of impact is directly related to how compatible the new contract version is with the old version and its surroundings in general.

This section establishes the fundamental types of compatibility that relate to the content and design of new contract versions and also tie into the goals and limitations of different versioning strategies introduced at the end of this appendix.

Backwards Compatibility

A new version of a service contract that continues to support consumer programs designed to work with the old version is considered *backwards-compatible*. From a design perspective, this means that the new contract has not changed in such a way that it can impact existing consumer programs that are already using the contract.

Backwards Compatibility in Web Services

A simple example of a backwards-compatible change is the addition of a new operation to an existing WSDL definition:

```
<definitions name="Purchase Order" targetNamespace=
  "http://actioncon.com/contract/po"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po"
  xmlns:po="http://actioncon.com/schema/po">
  ...
  <portType name="ptPurchaseOrder">
    <operation name="opSubmitOrder">
      <input message="tns:msgSubmitOrderRequest"/>
      <output message="tns:msgSubmitOrderResponse"/>
    </operation>
    <operation name="opCheckOrderStatus">
      <input message="tns:msgCheckOrderRequest"/>
      <output message="tns:msgCheckOrderResponse"/>
    </operation>
    <operation name="opChangeOrder">
      <input message="tns:msgChangeOrderRequest"/>
      <output message="tns:msgChangeOrderResponse"/>
    </operation>
    <operation name="opCancelOrder">
      <input message="tns:msgCancelOrderRequest"/>
      <output message="tns:msgCancelOrderResponse"/>
    </operation>
```

```
</operation>
<operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
</operation>
</portType>
</definitions>
```

Example F.2

The addition of a new operation represents a common backwards-compatible change.

NOTE

In this example we're borrowing the abstract description of the Purchase Order service that was initially built at the end of Chapter 7 (*Web Service Contract Design and Versioning for SOA*).

By adding a brand new operation, we are creating a new version of the contract, but this change is backwards-compatible and will not impact any existing consumers. The new service implementation will continue to work with old service consumers because all of the operations that an existing service consumer might invoke are still present and continue to meet the requirements of the previous service contract version.

Backwards Compatibility in REST Services

A backwards-compatible change to a REST-compliant service contract might involve adding some new resources or adding new capabilities to existing resources. In each of these cases the existing service consumers will only invoke the old methods on the old resources, which continue to work as they previously did.

```
Service: po.actioncon.com
Capabilities:
POST /orders
    In = application/vnd.com.actioncon.po+xml
GET /orders/{order-id}/status
    Out = text/plain
PUT /orders/{order-id}
    In = application/vnd.com/actioncon.po+xml
DELETE /orders/{order-id}
GET /orders/{order-id}
    Out = application/vnd.com.actioncon.po+xml
```

Example F.3

The addition of a new resource or new supported method on a resource is a backwards-compatible change for a REST service.

As demonstrated in Example F.3, supporting a new method that existing service consumers don't use results in a backwards-compatible change. However, in a service inventory with multiple REST services, we can take steps to ensure that new service consumers will continue to work with old versions of services.

As shown in Example F.4, it may be important for service consumers to have a reasonable way of proceeding with their interaction if the service reports that the new method is not implemented.

```
Legal methods for actioncon.com service inventory:  
* GET  
* PUT  
* DELETE  
* POST  
* SUBSCRIBE (consumers must fall back to periodic GET if service  
reports "not implemented")
```

Example F.4

New methods added to a service inventory's uniform contract need to provide a way for service consumers to "fall back" on a previously used method if they are to truly be backwards-compatible.

Changes to schemas and media types approach backwards compatibility in a different manner, in that they describe how information can be encoded for transport, and will often be used in both request and response messages. The focus for backwards compatibility is on whether a new message recipient can make sense of information sent by a legacy source. In other words, the new processor must continue to understand information produced by a legacy message *generator*.

An example of a change made to a schema for a message definition that is backwards-compatible is the addition of an optional element (as shown in bolded markup code in Example F.5).

```
Media type = application/vnd.com.actioncon.po+xml  
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
targetNamespace="http://actioncon.com/schema/po"  
xmlns="http://actioncon.com/schema/po">  
<xsd:element name="LineItem" type="LineItemType"/>  
<xsd:complexType name="LineItemType">  
    <xsd:sequence>  
        <xsd:element name="productID" type="xsd:string"/>  
        <xsd:element name="productName" type="xsd:string"/>  
        <b><xsd:element name="available" type="xsd:boolean"  
minOccurs="0"/>
```

```
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Example F.5

In an XML Schema definition, the addition of an optional element is also considered backwards-compatible.

Here we are using a simplified version of the XML Schema definition for the Purchase Order service. The optional `available` element is added to the `LineItemType` complex type. This has no impact on existing generators because they are not required to provide this element in their messages. New processors must be designed to cope without the new information if they are to remain backwards-compatible.

Changing any of the existing elements in the previous example from required to optional (by adding the `minOccurs="0"` setting) would also be considered a backwards-compatible change. When we have control over how we choose to design the next version of a Web service contract, backwards compatibility is generally attainable. However, mandatory changes (such as those imposed by laws or regulations) can often force us to break backwards compatibility.

NOTE

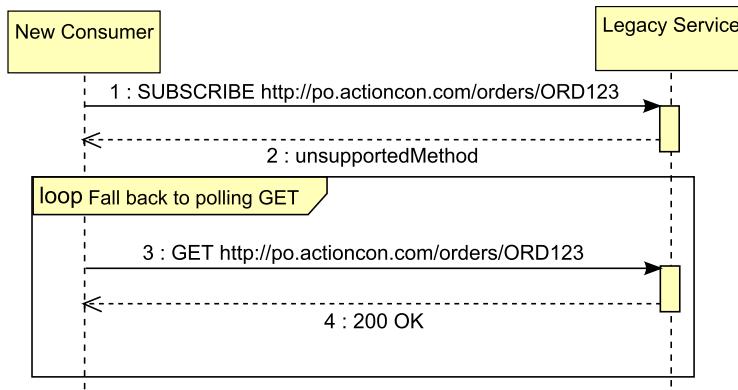
Both the Flexible and Loose versioning strategies explained at the end of this appendix support backwards compatibility.

Forwards Compatibility

When a service contract is designed in such a manner so that it can support a range of future consumer programs, it is considered to have an extent of *forwards compatibility*. This means that the contract can essentially accommodate how consumer programs will evolve over time.

Supporting forwards compatibility for Web service operations or uniform contract methods requires exception types to be present in the contract to allow service consumers to recover if they attempt to invoke a new and unsupported operation or method. For example, a “method not implemented” response enables the service consumer to detect that it is dealing with an incompatible service, thereby allowing it to handle this exception gracefully.

Redirection exception codes help REST services that implement a uniform contract change the resource identifiers in the contract when required. This is another way in which service contracts can allow legacy service consumers to continue using the service after contract changes have taken place.



Example F.6

A REST service ensures forwards compatibility by raising an exception whenever it does not understand a reusable contract or uniform contract method.

Forwards compatibility of schemas in REST services requires extension points to be present where new information can be added so that it will be safely ignored by legacy processors.

For example:

- any validation that the processor does must not reject a document formatted according to the new schema
- all existing information that the processor might need must remain present in future versions of the schema
- any new information added to the schema must be safe for legacy processors to ignore (if processors must understand the new information then the change cannot be forwards compatible)
- the processor must ignore any information that it does not understand

A common way to ensure validation does not reject future versions of the schema is to use wildcards in the earlier version. These provide extension points where new information can be added in future schema versions:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##any"/>
  </xsd:complexType>
</xsd:schema>
```

Example F.7

To support forwards compatibility within a message definition generally requires the use of XML Schema wildcards.

In this example, the `xsd:any` and `xsd:anyAttribute` elements are added to allow for a range of unknown elements and data to be accepted by the service contract. In other words, the schema is being designed in advance to accommodate unforeseen changes in the future.

NOTE

How wildcards can be used in support of forwards compatibility with Web services and the limited options that exist in support of forwards compatibility when it comes to WSDL definitions are discussed in Chapters 22 and 21 (*Web Service Contract Design and Versioning for SOA*), respectively.

It is important to understand that building extension points into service contracts for forwards compatibility by no means eliminates the need to consider compatibility issues when making contract changes. New information can only be added to schemas in a forwards compatible manner if it is genuinely safe for processors to ignore. New operations are only able to be made forwards compatible if a service consumer has an existing operation to fall back on when it finds the one it initially attempted to invoke is unsupported.

A service with a forwards compatible contract will often not be able to process all message content. Its contract is simply designed to accept a broader range of data unknown at the time of its design.

NOTE

Forwards compatibility forms the basis of the Loose versioning strategy that is explained shortly.

Compatible Changes

When we make a change to a service contract that does not negatively affect existing consumers, then the change itself is considered a *compatible change*.

NOTE

In this book, the term “compatible change” refers to backwards compatibility by default. When used in reference to forwards compatibility it is further qualified as a *forwards compatible change*.

A simple example of a compatible change is when we set the `minOccurs` attribute of an element from “1” to “0”, effectively turning a required element into an optional one, as shown here.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType"      <xsd:sequence>
    <xsd:element name="productID" type="xsd:string"/>
    <xsd:element name="productName" type="xsd:string"
      minOccurs="0"/>
    <xsd:element name="available" type="xsd:boolean"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Example F.8

The default value of the `minOccurs` attribute is “1”. Therefore because this attribute was previously absent from the `productName` element declaration, it was considered a required element. Adding the `minOccurs="0"` setting turns it into an optional element, resulting in a compatible change. (Note that making this change to a message output from the service would be an incompatible change.)

This type of change will not impact existing consumer programs that are used to sending the element value to the Web service, nor will it affect future consumers that can be designed to optionally send that element.

Another example of a compatible change was provided earlier in Example F.5, when we first added the optional `available` element declaration. Even though we extended the type with a whole new element, because it is optional it is considered a compatible change.

Here is a list of common compatible changes:

- adding a new WSDL operation definition and associated message definitions
- adding a new standard method to an existing REST resource
- adding a set of new REST resources
- changing the identifiers for a set of REST resources (including splitting and merging of services) using redirection response codes to facilitate migration of REST service consumers to the new identifiers
- adding a new WSDL port type definition and associated operation definitions
- adding new WSDL binding and service definitions
- extending an existing uniform contract method in a way that can be safely ignored by REST services that can fall back on old service logic (for example, adding “If-None-Match” as a feature of the HTTP GET operation so that if the service ignores it, the consumer will still get the current and correct representation for the resource)
- adding a new uniform contract method when an exception response exists for services that do not understand the method to use (and consumers can recover from this exception)
- adding a new optional XML Schema element or attribute declaration to a message definition
- reducing the constraint granularity of an XML Schema element or attribute of a message definition type used for input messages
- adding a new XML Schema wildcard to a message definition type
- adding a new optional WS-Policy assertion
- adding a new WS-Policy alternative

Other chapters in the *Web Service Contract Design and Versioning for SOA* book provide examples for several of these types of changes and further explore techniques whereby changes that are not normally compatible can still be implemented as compatible changes.

Incompatible Changes

If after a change a contract is no longer compatible with consumers, then it is considered to have received an *incompatible change*. These are the types of changes that can break an existing contract and therefore impose the most challenges when it comes to versioning.

NOTE

As with the term “compatible change,” this term also indicates backwards compatibility by default. When referring to incompatible changes that affect forwards compatibility, this term will be qualified as *forwards-incompatible change*.

Going back to our example, if we set an element’s `minOccurs` attribute from “0” to any number above zero, then we are introducing an incompatible change for input messages, as follows:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"
        minOccurs="3"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="3"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Example F.9

Incrementing the `minOccurs` attribute value of any established element declaration is automatically an incompatible change.

What was formerly an optional element is now required. This will certainly affect existing consumers that are not designed to comply with this new constraint, because adding a new required element introduces a mandatory constraint upon the contract.

Common incompatible changes include:

- renaming an existing WSDL operation definition
- removing an existing WSDL operation definition
- changing the MEP of an existing WSDL operation definition
- adding a fault message to an existing WSDL operation definition
- adding a new required XML Schema element or attribute declaration to a message definition
- increasing the constraint granularity of an XML Schema element or attribute declaration of a message definition
- renaming an optional or required XML Schema element or attribute in a message definition
- removing an optional or required XML Schema element or attribute or wildcard from a message definition
- adding a new required WS-Policy assertion or expression
- adding a new ignorable WS-Policy expression (most of the time)

Incompatible changes cause most of the challenges with service contract versioning. These and other types of incompatible changes are demonstrated in upcoming chapters (*Web Service Contract Design and Versioning for SOA*).

F.3 REST Service Compatibility Considerations

REST services within a given service inventory typically share a uniform contract for every resource, including uniform methods and media types. The same media types are used in both requests and responses, and new uniform contract facets are reused much more often than they are added to. This emphasis on service contract reuse within REST-compliant service inventories results in the need to highlight some special considerations, because changes to the uniform contract will automatically impact a range of service consumers because:

- the uniform contract methods are shared by all services
- the uniform contract media types are shared by both services and service consumers

As a result, both backwards compatibility and forwards compatibility considerations are almost equally important.

NOTE

Service contracts that make use of the Schema Centralization pattern without necessarily being REST-compliant will often need to impose a similarly rigid view of forward-compatibility and backwards compatibility.

Uniform contract methods codify the kinds of interactions that can occur between services and their consumers. For example, GET codifies “fetch some data,” while PUT codifies “store some data.”

Because the kinds of interactions that occur between REST services within the same service inventory tend to be relatively limited and stable, methods will usually change at a low rate compared to media types or resources. Compatibility issues usually pertain to a set of allowable methods that are only changed after careful case-by-case consideration.

An example of a compatible change to HTTP is the addition of `If-None-Match` headers to GET requests. If a service consumer knows the last version (or `etag`) of the resource that it fetched, it can make its GET request conditional. The `If-None-Match` header allows the consumer to state that the GET request should be shortcut and not executed if the version of the resource is still the same as it was for the consumer’s last fetch. If the service does not understand this new header, it will not shortcut its processing. Instead, it will return the normal GET response, although it will do so in a non-optimal mode.

An example of an incompatible change to HTTP is the addition of a `Host` header used to support multi-homing of Web servers. HTTP/1.0 did not require the name of the service to be included in request messages, but HTTP/1.1 does require this. If the special `Host` header is missing, HTTP/1.1 services must reject the request as being badly formed. However, HTTP/1.1 services are also required to be backwards-compatible, so if a HTTP/1.0 request comes into the REST service it will still be handled according to HTTP/1.0 rules.

Uniform contract media types further codify the kinds of information that can be exchanged between REST services and consumers. As previously stated, media types tend to change at a faster rate than HTTP methods in the uniform contract; however media types still change more slowly than resources. Compatible change is more of a live concern for the media types, and we can draw some more general rules about how to deal with them.

For example, if the generator of a message indicates to a processor of the message that it conforms to a particular media type, the processor generally does not need to know which version of the schema was used, nor does the processor need to have been built against the same version of the schema. The processor expects that all versions of the schema for a particular media type will be both forwards compatible and backwards-compatible with the type it was developed to support. Likewise, the generator expects that when it produces a message conformant with a particular schema version, that all processors of the message will understand it.

When incompatible changes are made to a schema, a new media type identifier is generally required to ensure that:

- the processor can decide how to parse a document based on the media type identifier
- services and consumers are able to *negotiate* for a specific media type that will be understood by the processor when the message has been produced

Content negotiation is the ultimate fall-back to ensure compatibility in REST-compliant service inventories. For a fetch interaction this often involves the consumer indicating to the service what media types it is able to support, and the service returning the most appropriate type that it supports. This mechanism allows for incompatible changes to be made to media types, as required.

NOTE

One way to better understand versioning issues that pertain to media types is to look at how they are used in HTML. An example of a compatible change to HTML that did not result in the need for a new media type was the addition of the abbr element to version 4.0 of the HTML language. This element allows new processors of HTML documents to support a mouse-over to expand abbreviations on a web page and to better support accessibility of the page. Legacy processors safely ignore the expansion, but will continue correctly showing the abbreviation itself.

An example of an incompatible change to HTML that did require a new media type was the conversion of HTML 4.0 to XML (resulting in version 1.0 of XHTML). The media type for the traditional SGML version remained `text/html`, while the XML version became `application/xhtml+xml`. This allowed content negotiation to occur between the two types, and for processors to choose the correct parser and validation strategy based on which type was specified by the service.

Some incompatible changes have also been made to HTML without changing the media type. HTML 4.0 deprecated APPLET, BASEFONT, CENTER, DIR, FONT, ISINDEX, MENU, S, STRIKE, and U elements in favor of newer elements. These elements must continue to be understood but their use in HTML documents is being phased out. HTML 4.0 made LISTING, PLAINTEXT, and XMP obsolete. These elements should not be used in HTML 4.0 documents and no longer need to be understood.

Deprecating elements over a long period of time and eventually identifying them as obsolete once they are no longer used by existing services or consumers is a technique that can be used for REST media types to incrementally update a schema without having to change the media type.

F.4 Version Identifiers

One of the most fundamental design patterns related to Web service contract design is the Version Identification pattern. It essentially advocates that version numbers should be clearly expressed, not just at the contract level, but right down to the versions of the schemas that underlie the message definitions.

The first step to establishing an effective versioning strategy is to decide on a common means by which versions themselves are identified and represented within Web service contracts.

Versions are almost always communicated with version numbers. The most common format is a decimal, followed by a period and then another decimal, as shown here:

```
version="2.0"
```

Sometimes, you will see additional period + decimal pairs that lead to more detailed version numbers like this:

```
version="2.0.1.1"
```

The typical meaning associated with these numbers is the measure or significance of the change. Incrementing the first decimal generally indicates a major version change (or upgrade) in the software, whereas decimals after the first period usually represent various levels of minor version changes.

From a compatibility perspective, we can associate additional meaning to these numbers. Specifically, the following convention has emerged in the industry:

- A minor version is expected to be backwards-compatible with other minor versions associated with a major version. For example, version 5.2 of a program should be fully backwards-compatible with versions 5.0 and 5.1.
- A major version is generally expected to break backwards compatibility with programs that belong to other major versions. This means that program version 5.0 is not expected to be backwards-compatible with version 4.0.

NOTE

A third “patch” version number is also sometimes used to express changes that are both forwards-compatible and backwards-compatible. Typically these versions are intended to clarify the schema only, or to fix problems with the schema that were discovered once it was deployed. For example, version 5.2.1 is expected to be fully compatible with version 5.2.0, but may be added for clarification purposes.

This convention of indicating compatibility through major and minor version numbers is referred to as the *compatibility guarantee*. Another approach, known as “amount of work,” uses version numbers to communicate the effort that has gone into the change. A minor version increase indicates a modest effort, and a major version increase predictably represents a lot of work.

These two conventions can be combined and often are. The result is often that version numbers continue to communicate compatibility as explained earlier, but they sometimes increment by several digits, depending on the amount of effort that went into each version.

There are various syntax options available to express version numbers. For example, you may have noticed that the declaration statement that begins an XML document can contain a number that expresses the version of the XML specification being used:

```
<?xml version="1.0"?>
```

That same `version` attribute can be used with the root `xsd:schema` element, as follows:

```
<xsd:schema version="2.0" ...>
```

You can further create a custom variation of this attribute by assigning it to any element you define (in which case you are not required to name the attribute “`version`”).

```
<LineItem version="2.0">
```

An alternative custom approach is to embed the major version number into a namespace or media type identifier, as shown here:

```
<LineItem xmlns="http://actioncon.com/schema/po/v2">
```

or

```
application/vnd.com.actioncon.po.v2+xml
```

Note that it has become a common convention to use date values in namespaces when versioning XML schemas, as follows:

```
<LineItem xmlns="http://actioncon.com/schema/po/2010/09">
```

In this case, it is the date of the change that acts as the major version identifier. In order to keep the expression of XML Schema definition versions in alignment with WSDL definition versions, we use version numbers instead of date values in upcoming examples. However, when working in an environment where XML Schema definitions are separately owned as part of an independent data architecture, it is not uncommon for schema versioning identifiers to be different from those used by WSDL definitions.

Regardless of which option you choose, it is important to consider the Canonical Versioning pattern that dictates that the expression of version information must be standardized across all service contracts within the boundary of a service inventory. In larger environments, this will often require a central authority that can guarantee the linearity, consistency, and description quality of version information. These types of conventions carry over into how service termination information is expressed, as further explored in Chapter 23 (*Web Service Contract Design and Versioning for SOA*).

NOTE

Of course you may also be required to work with third-party schemas and WSDL definitions that may already have implemented their own versioning conventions. In this case, the extent to which the Canonical Versioning pattern can be applied will be limited.

F.5 Versioning Strategies

There is no one versioning approach that is right for everyone. Because versioning represents a governance-related phase in the overall lifecycle of a service, it is a practice that is subject to the conventions, preferences, and requirements that are distinct to any enterprise.

Even though there is no de facto versioning technique for the WSDL, XML Schema, and WS-Policy content that comprises Web service contracts, a number of common and advocated versioning approaches have emerged, each with its own benefits and tradeoffs.

Here we are going to single out the following three common strategies:

- *Strict* – Any compatible or incompatible changes result in a new version of the service contract. This approach does not support backwards or forwards compatibility.
- *Flexible* – Any incompatible change results in a new version of the service contract and the contract is designed to support backwards compatibility but not forwards compatibility.
- *Loose* – Any incompatible change results in a new version of the service contract and the contract is designed to support backwards compatibility and forwards compatibility.

These strategies are explained individually in the upcoming sections.

The Strict Strategy (New Change, New Contract)

The simplest approach to Web service contract versioning is to require that a new version of a contract be issued whenever any kind of change is made to any part of the contract.

This is commonly implemented by changing the target namespace value of a WSDL definition (and possibly the XML Schema definition) every time a compatible or incompatible change is made to the WSDL, XML Schema, or WS-Policy content related to the contract. Namespaces are used for version identification instead of a version attribute because changing the namespace value automatically forces a change in all consumer programs that need to access the new version of the schema that defines the message types.

This “super-strict” approach is not really that practical, but it is the safest and sometimes warranted when there are legal implications to Web service contract modifications, such as when contracts are published for certain inter-organization data exchanges. Because both compatible and incompatible changes will result in a new contract version, this approach supports neither backwards or forwards compatibility.

Pros and Cons

The benefit of this strategy is that you have full control over the evolution of the service contract, and because backwards and forwards compatibility are intentionally disregarded, you do not need to concern yourself with the impact of any change in particular (because all changes effectively break the contract).

On the downside, by forcing a new namespace upon the contract with each change, you are guaranteeing that all existing service consumers will no longer be compatible with any new version of the contract. Consumers will only be able to continue communicating with the Web service while the old contract remains available alongside the new version or until the consumers themselves are updated to conform to the new contract.

Therefore, this approach will increase the governance burden of individual services and will require careful transitioning strategies. Having two or more versions of the same service co-exist at the same time can become a common requirement for which the supporting service inventory infrastructure needs to be prepared.

The Flexible Strategy (Backwards Compatibility)

A common approach used to balance practical considerations with an attempt at minimizing the impact of changes to Web service contracts is to allow compatible changes to occur without forcing a new contract version, while not attempting to support forwards compatibility at all.

This means that any backwards-compatible change is considered safe in that it ends up extending or augmenting an established contract without affecting any of the service’s

existing consumers. A common example of this is adding a new operation to a WSDL definition or adding an optional element declaration to a message's schema definition.

As with the Strict strategy, any change that breaks the existing contract does result in a new contract version, usually implemented by changing the target namespace value of the WSDL definition and potentially also the XML Schema definition.

Pros and Cons

The primary advantage to this approach is that it can be used to accommodate a variety of changes while consistently retaining the contract's backwards compatibility. However, when compatible changes are made, these changes become permanent and cannot be reversed without introducing an incompatible change. Therefore, a governance process is required during which each proposed change is evaluated so that contracts do not become overly bloated or convoluted. This is an especially important consideration for agnostic services that are heavily reused.

The Loose Strategy (Backwards and Forwards Compatibility)

As with the previous two approaches, this strategy requires that incompatible changes result in a new service contract version. The difference here is in how service contracts are initially designed.

Instead of accommodating known data exchange requirements, special features from the WSDL, XML Schema, and WS-Policy languages are used to make parts of the contract intrinsically extensible so that they remain able to support a broad range of future, unknown data exchange requirements.

For example:

- The `anyType` attribute value provided by the WSDL 2.0 language allows a message to consist of any valid XML document.
- XML Schema wildcards can be used to allow a range of unknown data to be passed in message definitions.
- Ignorable policy assertions can be defined to communicate service characteristics that can optionally be acknowledged by future consumers.

These and other features related to forwards compatibility are discussed in upcoming chapters (*Web Service Contract Design and Versioning for SOA*).

Pros and Cons

The fact that wildcards allow undefined content to be passed through Web service contracts provides a constant opportunity to further expand the range of acceptable message element and data content. On the other hand, the use of wildcards will naturally result in vague and overly coarse service contracts that place the burden of validation on the underlying service logic.

Summary Table

Provided here is a table that broadly summarizes how the three strategies compare based on three fundamental characteristics.

	Strategy		
	Strict	Flexible	Loose
Strictness	high	medium	low
Governance Impact	high	medium	high
Complexity	low	medium	high

Table F.1

A general comparison of the three versioning strategies.

The three characteristics used in this table to form the basis of this comparison are as follows:

- *Strictness* – The rigidity of the contract versioning options. The Strict approach clearly is the most rigid in its versioning rules, while the Loose strategy provides the broadest range of versioning options due to its reliance on wildcards.
- *Governance Impact* – The amount of governance burden imposed by a strategy. Both Strict and Loose approaches increase governance impact but for different reasons. The Strict strategy requires the issuance of more new contract versions, which impacts surrounding consumers and infrastructure, while the Loose approach introduces the concept of unknown message sets that need to be separately accommodated through custom programming.

- *Complexity* – The overall complexity of the versioning process. Due to the use of wildcards and unknown message data, the Loose strategy has the highest complexity potential, while the straight-forward rules that form the basis of the Strict approach make it the simplest option.

Throughout this comparison, the Flexible strategy provides an approach that represents a consistently average level of strictness, governance effort, and overall complexity.

F.6 REST Service Versioning Considerations

REST services that share the same uniform contract maintain separate versioned specifications for the following:

- the version number or specification of the resource identifier syntax (as per the “Request for Comments 6986 - Uniform Resource Identifier (URI): Generic Syntax” specification)
- the specification of the collection of legal methods, status codes, and other interaction protocol details (as per the “Request for Comments 2616 - Hypertext Transfer Protocol - HTTP/1.1” specification)
- individual specifications for legal media types (for example, HTML 4.01 and the “Request for Comments 4287 - The Atom Syndication Format” specification)
- individual specifications for service contracts that use the legal resource identifier syntax, methods, and media types

Each part of the uniform contract is specified and versioned independently of the others. Changing any one specification does not generally require another specification to be updated or versioned. Likewise, changing any of the uniform contract facet specifications does not require changes to individual service contracts, or changes to their version numbers.

This last point is in contradiction to some conventional versioning strategies. One might expect that if a schema used in a service contract changed, then the service contract would need to be modified. However, with REST services there is a tendency to maintain both forwards compatibility and backwards compatibility. If a REST service consumer sends a message that conforms to a newer schema, the service can process it as if it conformed to the older schema. If compatibility between these schemas has been maintained then the service will function correctly. Likewise, if the service returns a

message to the consumer that conforms to an old schema the newer service consumer can still process the message correctly.

REST service contracts only need to directly consider the versioning of the uniform contract when media types used become deprecated, or when the schema advances so far that elements and attributes the service depends on are on their way to becoming obsolete. When this occurs, the service contract needs to be updated, and with it, the underlying service logic that processes the media types.

Appendix G



Mapping Service-Orientation to RUP

The Rational Unified Process (RUP) is commonly used for projects with identified areas of complexity and risk, and that can involve numerous potential stakeholders. RUP has been successfully applied in many organizations; today, a large number of RUP practitioners use a growing library of best practices to guide software development projects.

This appendix is intended for IT professionals with a RUP background who want to better understand how RUP can apply or relate to SOA projects, as well as those with an SOA background required to work with RUP. The upcoming sections will present an overview of how RUP can relate to service-orientation, SOA and MSOAM, the Mainstream SOA Methodology [REF-3]. We will explore how key RUP principles align with the goals, benefits, and pillars introduced in Chapters 3 and 4 of this book. We then further provide a high level mapping of service-orientation to RUP phases, milestones, disciplines, roles, activities and artifacts.

The purpose of this mapping is to determine:

- gaps between RUP and service-orientation
- how RUP emphasizes particular elements of SOA and service-orientation
- how service-orientation emphasizes particular aspects of RUP
- how RUP can be extended with service-orientation elements
- how the application of service-orientation can benefit from RUP practices

Compatibility of RUP and SOA

The findings from “Suitability of Extreme Programming and RUP Software Development Methodologies for SOA Applications” [REF-1] provides an abstract mapping between RUP and SOA methodologies. Five criteria were used to demonstrate how RUP can support service delivery projects (Table G.1).

Characteristic	SOA (requires)	RUP (supports)
Size of Development Team	multidisciplinary teams	multidisciplinary teams
Level of Documentation	very high	high
Development Time	long when first designing the architecture, short when the architecture is in place	long, but steady development
Scope of Software	enterprise, domain	any scope
Type of Orientation	service	object

Table G.1

Five process characteristics required by SOA and supported by RUP.

RUP supports four of the listed characteristics, but does not support the fifth (type of orientation). The suggestion is that RUP needs can support service-oriented architecture development by incorporating SOA methodology to add a layer on top of the object-oriented design features.

This paper and other previous work on mapping or combining RUP with SOA concentrated mainly on introducing activities specific to SOA into RUP, or partially distributing basic SOA project stages into RUP phases.

When mapping aspects of service-orientation to RUP, further alternatives come to light. The main building blocks, principles, and best practices of RUP and MSOAM can be combined to elaborate on the relationships and gaps between the two methodologies. That is, we can establish a mapping among all areas of MSOAM related to service delivery and all the engineering disciplines of RUP with the associated processes, roles, artifacts, activities, and guidelines. When gaps are identified, we can decide to extend one process by introducing elements of the other. When mapping is established, the processes can leverage features, patterns, and best practices from each other.

Overview of RUP (and MSOAM)

The main building blocks of RUP, or content elements, are the following:

- Roles (who) – A Role defines a set of related skills, competencies and responsibilities.

- Artifacts/Work Products (what) – A work product represents something resulting from a task, including all the documents and models (artifacts) produced while working through the process.
- Activities/Tasks (how) – A task describes a unit of work assigned to a Role that provides a meaningful result.

The above are organized into phases, iterations, disciplines, and workflow details (when). Within each iteration, the tasks are categorized into nine disciplines:

- six engineering disciplines: business modeling, requirements, analysis and design, implementation, test, deployment
- three supporting disciplines: configuration and change management, project management, environment

RUP is architecture-centric and risk-driven in that values are produced by the engineering disciplines. MSOAM is business-driven and enterprise-centric and constantly targets the delivery of value to the business, regardless of how the business changes.

The upcoming sections focus on the engineering disciplines of RUP and associated roles, along with the project delivery stages of MSOAM and corresponding roles.

The Pillars of Service-Orientation and the RUP Principles

The four pillars of service-orientation (introduced in Chapter 4) are summarized in Table G.2.

Pillar	Description
Teamwork	Cross-project and cross-functional teams and cooperation are required.
Education	Team members must communicate and cooperate based on common knowledge (common vocabulary, definitions, concepts, methods) and understanding of the target state the team is collectively working to attain.
Discipline	Consistent application of common knowledge by all members of the teams.
Balanced Scope	Establish teamwork, education, and discipline within a meaningful and manageable scope.

Table G.2

The four SOA Pillars.

One of the key RUP principles (known as “ABCDEF”) is entirely devoted to Teamwork and collaboration. Furthermore, RUP supports education and common understanding in the following ways:

- by providing training and mentoring as activities [REF-8]
- by using multiple levels of abstraction with well defined concepts and methods
- by using UML as a graphical language

The pillar of discipline is supported by RUP’s six best practices, while the key principles contribute toward the attainment of a balanced scope. Table G.3 elaborates on this comparison.

Pillar	RUP Support
Teamwork	<p>Key principle - Collaborate across teams:</p> <ul style="list-style-type: none"> • Motivate people to perform at their best. • Collaborate cross-functionally across analysts, developers, testers. • Manage evolving artifacts and tasks to enhance collaboration and progress/quality insight with integrated environments. • Ensure that business, development, and operations teams work effectively as an integrated whole.
Education	<p>Key principle - Elevate the level of abstraction:</p> <ul style="list-style-type: none"> • Reuse existing assets • Reduce the amount of human-generated stuff through higher-level tools and languages. • Architect for resilience, quality, understandability, and complexity control. <p>Use UML as standard language in analysis and design.</p> <p>Training and mentoring are further supported activities throughout practices that introduce RUP into organizations.</p>

continues

Pillar	RUP Support
Discipline	<p>Systematic, methodical way to analyze, design, develop, and validate with the six best practices:</p> <ol style="list-style-type: none"> 1. Develop software iteratively. 2. Manage requirements. 3. Use component-based architectures. 4. Visually model software. 5. Verify software quality. 6. Control changes to software.
Balanced Scope	<p>Key principle – Adapt the process.</p> <p>Key principle – Demonstrate value iteratively.</p>

Table G.3

How RUP supports the four pillars of service-orientation.

It should be noted here that although the service-orientation pillars correspond to RUP principles, activities, and roles, RUP still needs to be extended in the following areas:

- The collaboration across teams requires additional attention due to the potential complexity of incorporating additional enterprise and governance roles within the project.
- Educational materials need to be extended with the eight service-orientation principles, patterns that address service analysis and service design, and precepts and processes specific to shared service governance.

Note that the business vocabulary is optional in RUP but mandatory in SOA.

There are two further RUP key principles that have not been mapped:

- Balance competing stakeholder priorities.
- Focus continuously on quality.

These can contribute to the successful execution of several service delivery stages in MSOAM.

Breadth and Depth Roles and Role Mapping

RUP role definitions are consistent with the notion of separating *breadth and depth* [REF-7]. As the name implies, depth roles perform tasks that add details and precision to the artifacts produced by the breadth roles. Breadth roles are related to leadership, impact and integrity, definition of standards, and guidelines. The breadth and depth are not related to the discipline. Each of RUP's nine disciplines has one role that focuses on breadth, and a different role that focuses on depth for that discipline.

Furthermore, roles can be grouped in role sets. There are four role sets: Analysts, Developers, Testers and Managers. Once this basic principle is understood, it is easy to classify the roles in meaningful groups and map them to MSOAM roles (Table G.4).

MSOAM Project Stage	MSOAM Role	RUP Role Set	RUP Discipline	RUP Breadth Role	RUP Depth Role
Service Inventory Analysis	Business Analyst	Analyst	Business Modeling	Business Process Analyst	Business Designer
Service-Oriented Analysis	Service Analyst	Analyst	Requirements	Systems Analyst	Requirements Specifier
Service-Oriented Design Service Logic Design	Service Architect	Developer	Analysis and Design	Software Architect	Designer
Service Development	Service Developer	Developer	Implementation	Integrator	Implementer
Service Testing	SOA Quality Assurance Specialist	Manager/ Analyst/ Developer/ Tester	Test	Test Manager Test Analyst Test Designer	Test Designer Tester

continues

MSOAM Project Stage	MSOAM Role	RUP Role Set	RUP Discipline	RUP Breadth Role	RUP Depth Role
Service Deployment and Maintenance	Service Administrator	Manager	Deployment	Deployment Manager	Tech Writer, Course Developer, Graphic Artist
All	IT Manager	Manager	Project Management	Project Manager	Project Manager

Table G.4

MSOAM roles and project stages mapped to RUP breadth/depth roles, disciplines, and role sets.

It is important to note that there is no distinction between breadth and depth for the above MSOAM roles. However, in larger programs it will become necessary to make this distinction, especially in relation to Service Analysts where the concept of *Lead Analyst* emerges.

One further key observation is the gravity that testing has in RUP. The Test Discipline includes roles from all the role sets: Managers, Analysts, Developers, and Testers. This fully supports the importance placed upon the Testing stage within SOA delivery projects (where services need to be tested rigorously as if they were commercial products).

Enterprise and Governance Roles

Parallel to the traditional roles, MSOAM introduces:

- Four enterprise or domain level custodian roles: Schema Custodian, Policy Custodian, Service Registry Custodian, Enterprise Design Standards Custodian (and Auditor). These roles are active mainly during the Service-Oriented Design, Service Logic Design, Service Deployment and Maintenance, and Service Versioning and Retirement stages.
- One generic enterprise architecture role: Enterprise Architect. This role is present in just about all stages of an SOA delivery project, but the only project-specific artifact directly produced is the inventory blueprint.
- Two service management roles: Service Custodian, Service Administrator
- One general governance role: SOA Governance Specialist

All of these roles are responsible in defining some form of standards and participate in reviews. Additionally, MSOAM introduces the concept of the SOA Governance Program Office, which has a scope well beyond that of the Change Control Board established during the Inception phase in RUP.

Mapping Service Delivery Project Stages to Disciplines

MSOAM defines eleven project lifecycle stages, nine of which are service delivery stages. Only the service delivery stages are relevant for mapping to RUP (Table G.5), because RUP does not support generic product lifecycle stages.

Service Delivery Project Stages	RUP Disciplines
Service Inventory Analysis Service-Oriented Analysis Service-Oriented Design Service Logic Design	Business Modeling Requirement Analysis and Design
Service Development	Implementation
Service Testing	Test
Service Deployment and Maintenance	Deployment
—	Project Management
—	Environment
Service Discovery	—
Service Versioning and Retirement	Configuration and Change Management

Table G.5

Mapping of MSOAM service delivery stages with RUP disciplines.

The service delivery stages are further divided into sub-processes or activities. This subdivision allows us to make the mapping between the Service Inventory Analysis, Service-Oriented Analysis, and Service-Oriented Design stages more granular with the corresponding RUP disciplines.

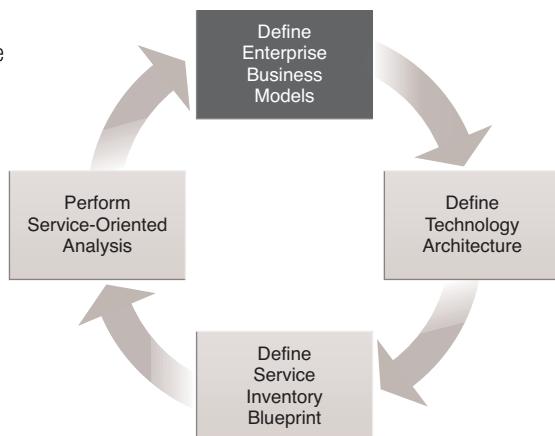
As mentioned previously, we need to be careful when applying this level of mapping; even if we assume that the roles, activities, and artifacts correspond entirely, we still must consider the gaps associated by the followed principles, patterns, and methods. (These gaps are briefly discussed in the upcoming *Service-Orientation and RUP: Gaps* section.)

Mapping MSOAM Analysis and Design Stages to RUP Disciplines

In the top-down project approach, the Service Inventory Analysis lifecycle consists of four processes that are applied iteratively (Figure G.1). These processes can be mapped to three different RUP disciplines (Table G.6).

Figure G.1

The four primary steps that comprise the Service Inventory Analysis stage.



The Define Enterprise Business Models process corresponds to RUP Business Modeling. It is the process in which all business concepts, terms, and architecture are defined and modeled. It is probably the only mapping between MSOAM and RUP which is completely free of gaps. That is, one can safely apply the Business Modeling of RUP to the Service Inventory Analysis lifecycle.

The mapping of the Define Technology Architecture process to the RUP Analysis and Design discipline is also straightforward. What may appear odd at first is that the Analysis and Design seems to be applied together with Business Modeling without any mention of the Requirements discipline. However, this is not an issue because in RUP there is no sequence of disciplines; only sequences of iterations and phases. All principles can be combined together in the same iteration. (That is, the three disciplines that appear in Table G.6 can be applied simultaneously in each iteration in the inception and elaboration phase.)

Service Inventory Analysis Process	MSOAM Role	RUP Discipline	RUP Breadth Role	Primary RUP Artifacts
Define Enterprise Business Models	Business Analyst	Business Modeling	Business Process Analyst	Business Vision Business Architecture Business Glossary Suppl. Business Specification Business UCM Business Object Model
Define Technology Architecture	Service Architect	Analysis and Design	Software Architect	Software Architecture Document (Inception)
Define Service Inventory Blueprint	Service Analyst	Requirement (Workflow: Define the System)	System Analyst	Vision Glossary UCM Supplemental Specification
Perform Service-Oriented Analysis	Service Analyst	Requirement	System Analyst	Vision Glossary UCM Supplemental Specification

Table G.6

Mapping Service Inventory Analysis processes and roles to RUP disciplines, roles, and artifacts.

The mapping of the Definition of the Service Inventory Blueprint process to the Requirement RUP discipline deserves some further elaboration: The primary role assigned to the Definition of the Service Inventory Blueprint process is the Service Analyst. This role corresponds to the System Analyst in RUP, and is the breadth role for the Requirement discipline. Furthermore, the service inventory blueprint represents a collection of conceptual service candidates, and is therefore not restricted to the service inventory architecture. This aligns very well with the Define System Workflow process in RUP, which is part of the workflow of the Requirement discipline. The artifacts processed by this Workflow include the Use Case Model, the Vision, and the Object Model, which, when combined, correspond to the modeling of service candidates.

Service-Orientation and RUP: Gaps

As a result of the mapping performed, some notable gaps were identified. For example, the four characteristics of service-oriented technology architecture (vendor-neutral, business-driven, enterprise-centric, composition-centric) are not present in RUP. Furthermore, core service-orientation concepts, like services and service compositions, replace the RUP concepts of objects and object-oriented applications. Other observations include the fact that service-oriented solutions place a greater emphasis on security, standardization, centralization, and cross-project collaboration.

The proper application of the pillars of service-orientation will help organizations overcome these gaps.

Related Reading

It is worth acknowledging other work that has been done in the area of mapping SOA with RUP. For example:

- A RUP for SOA plug-in was made available in 2005 to help extend the RUP analysis and design approach with workflows and artifacts required for service-oriented analysis and design.
- A mapping of high level service design stages to RUP phases was published in “Mapping of SOA and RUP” [REF-5], but this work did not cover service-oriented analysis.
- A combination of RUP and the MSOAM appeared in the March 2008 issue of the *SOA Magazine* [REF-4]. This article uses RUP to elaborate service inventories and it focuses on processes related to top-down Service-Oriented Analysis stages in relation to the RUP activities related to Business Modeling.
- The “Enterprise Unified Process” [REF-6] delves into extending RUP with activities related to SOA or enterprise architecture in general. The limitations of RUP for operation and support of software after its deployment are treated by the Enterprise Unified Process (EUP), an extension of RUP.

Bibliography

- [REF-1] Suitability of Extreme Programming and RUP Software Development Methodologies for SOA Applications, Guillermo A. Callahan, 2006 (www.soberit.hut.fi/T-86/T-86.5165/2007/final_callahan.pdf).
- [REF-2] *The Rational Unified Process: an Introduction*, P. Kruchten, Addison Wesley, 2000.
- [REF-3] “MSOAM, The Mainstream SOA Methodology,” Thomas Erl(www.soamethodology.com).
- [REF-4] “Working with SOA and RUP,” Solmaz Boroumand, *SOA Magazine*, Issue XVI, March 2008.
- [REF-5] “Mapping of SOA and RUP: DOA as Case Study,” Shahid Hussain, Bashir Ahmad, Shakeel Ahmad, Sheikh Muhammad Saqib, *Journal of Computing*, January 2010.
- [REF-6] *Enterprise Unified Process: Extending the Rational Unified Process*, Scott W. Ambler, John Nalbone, and Michael Vizdos, Prentice Hall (www.enterpriseunifiedprocess.com).
- [REF-7] *Understanding RUP Roles*, Anthony Crain, IBM (www.ibm.com/developerworks/rational/library/apr05/crain/index.html).
- [REF-8] “Introducing the RUP into an organization,” DJ de Villiers, 2003(www.ibm.com/developerworks/rational/library/916.html)
- [REF-9] “Rational Unified Process: Best Practices for Software Development Teams,” Rational Software White Paper (www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf).

This page intentionally left blank



Appendix H

Additional Resources

Following is a list of resource Web sites that provide supplementary content for books in this series. If you'd like to be automatically notified of new book releases, new supplementary content for this title, or key changes to these Web sites, send a blank e-mail to notify@soabooks.com.

www.soabooks.com

The official site of the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*. Numerous resources are provided, including sample chapters from available books and updates and corrections.

www.soaschool.comwww.cloudschool.com

These two educational sites describe the vast curricula dedicated to SOA and Cloud Computing. Books from the *Prentice Hall Service-Oriented Computing Series from Thomas Erl* are official parts of these programs. In particular, SOASchool.com offers a series of courses for the Certified SOA Governance track for which this book is a primary resource.

www.soamag.com

This site is the home of The SOA Magazine, a monthly publication officially associated with this book series. This magazine is dedicated to publishing specialized articles, case studies, and papers that explore various aspects of service-oriented computing.

www.soaglossary.com

A master glossary for all books in the *Prentice Hall Service-Oriented Computing Series by Thomas Erl* is hosted by this site. This site is constantly growing as new titles are developed and released.

www.soaspecs.com

This Web site establishes a convenient central portal to industry standards and specifications covered or referenced by titles in this book series.

www.soapatterns.org

The official site of the master catalog of SOA design patterns. This site allows for the online submission and community review of candidate patterns proposed for inclusion in the master patterns catalog.

www.serviceorientation.com, www.whatissoa.com, www.whatiscloud.com

These sites provide papers, book excerpts, and various content dedicated to describing and defining the service-orientation paradigm, cloud computing concepts, associated principles, and the service-oriented technology architectural model.

www.soasymposium.com/www.cloudsymposium.com

Two sites dedicated to the conference series featuring the world's largest SOA and Cloud Computing events. These co-located conferences are held throughout the world and frequently feature authors from the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*.

This page intentionally left blank

About the Authors

Stephen G. Bennett

Stephen G. Bennett currently holds the role of Senior Enterprise Architect at Oracle, prior to which he worked with BEA where he was the Americas SOA Practice Lead within BEA's consulting division. Stephen is a 25-year experienced manager and technologist, with a wide range of leadership, architecture, and implementation experience around SOA and Cloud Computing gained in high profile environments. Before becoming a consultant, Stephen spent 12 years in the investment banking industry delivering global trading systems.

Alongside many white papers and magazine articles, Stephen's previous literary efforts include the book *Silver Clouds, Dark Linings: A Concise Guide to Cloud Computing* (Prentice Hall 2010). Stephen is a regular speaker at executive events and conferences on topics such as SOA adoption, service engineering, SOA Governance, service-oriented architecture, and cloud computing. Stephen has been involved in multiple standards efforts around SOA and Enterprise Architecture. Stephen has co-chaired a number of working groups within the Open Group organization around SOA Governance and TOGAF/SOA.

Thomas Erl

Thomas Erl is the founder of SOASchool.com® and CloudSchool.com™, as part of Arcitura Education Inc. Thomas has been the world's top-selling SOA author for more than five years and is the series editor of the *Prentice Hall Service-Oriented Computing Series from Thomas Erl*, as well as the editor of the *SOA Magazine*. With more than 140,000 copies in print world-wide, his seven published books have become international best-sellers and have been formally endorsed by senior members of major IT organizations, such as IBM, Microsoft, Oracle, Intel, Accenture, IEEE, MITRE, SAP, CISCO, and HP.

In cooperation with SOASchool.com® and CloudSchool.com™, Thomas has helped develop curricula for the internationally recognized SOA Certified Professional (SOACP) and Cloud Certified Professional (CCP) accreditation programs, which have established a series of formal, vendor-neutral industry certifications.

Thomas is the founding member of the SOA Manifesto Working Group (www.soamanifesto.org), founder of the APQC Service-Orientation Maturity Model (SOMM) initiative, co-chair of the SOA Education Committee, and he further oversees the SOAPatterns.org initiative, a community site dedicated to the on-going development of a master patterns catalog for service-oriented computing.

Thomas has toured more than 20 countries as a speaker and instructor for public and private events, and regularly participates in SOA Symposium (www.soasymposium.com) and Gartner conferences. More than 100 articles and interviews by Thomas have been published in numerous publications, including the *Wall Street Journal* and *CIO Magazine*.

Clive Gee, Ph.D.

After developing an interest in computers while studying for a Ph.D. in Theoretical Physics from the University of Stirling, Scotland, Clive joined IBM United Kingdom in 1976 to pursue a career in the emerging IT industry. He worked initially in telecommunications and office automation, and then moved to the field of application development in the 1980s where he spent the remainder of his career.

An early proponent of Object Orientation, he was one of the founders of IBM's European Object Technology Practice, where he worked on major client application development projects and internal CASE tool development. In 1997 Clive moved to the United States, joining IBM's North American Object Technology Practice as a consultant architect, working on major client projects in the Banking, Retail, Telecommunication, and Transportation Industries. During his tenure with IBM he worked on developing solutions that ranged from wireless telecommunications network infrastructure to mobile applications for the airline industry.

As well as being closely involved in the technical architecture and design of complex IT solutions, Clive developed an interest in the field of software engineering, improving the IT application design and development process by adopting production management techniques such as those used by the engineering and manufacturing industries. One of the very first IBM architects to work on SOA, Clive was involved with most of IBM's flagship SOA engagements, initially as a Solution or Lead Architect, then increasingly

as a specialist in SOA governance, where he is considered to be one of IBM's pre-eminent worldwide practitioners. Clive worked on numerous major client projects in the USA, Canada, Latin America, Europe, Japan, and Australia. Semi-retired in 2008, Clive has since returned to live in the UK's Northern Isles. He is a co-author of the book *SOA Governance: Achieving and Sustaining Business and IT Agility* (IBM Press 2008).

Robert Laird

Robert is the lead architect of the IBM Software Group in areas of SOA governance and SOA policy; he currently leads the automation of the SOA Policy Lifecycle. Prior to that, Robert co-authored the *SOA Governance and Management Method* (SGMM) for usage of SOA governance capabilities and maturity assessment. Robert has several years of international consulting experience and was responsible for supporting and leading service-oriented architecture (SOA) governance and SOA architecture engagements for worldwide IBM customers.

With more than 20 years experience in the telecom industry at MCI and Verizon, Robert has been the MCI chief architect, leading the enterprise architecture group and has worked across the entire order-to-cash suite of applications. He led the development of the SOA based single stack strategy to simplify the multiple network and applications silos; he has driven the strategy, planning, and execution of MCI's product development in the area of contact centers, IP/VPN, VOIP, IMS, and managed services; and, for OSS, he has led successful implementations to automate network provisioning, network restoration, and network management.

Prior to joining MCI, Robert worked as a consultant for American Management Systems (AMS) and Ideation, Inc. He has an MS and a BS degree in Computer Science from Purdue University and has been granted two patents in the area of telephony, with three patents pending in the area of computing. As well as speaking at various industry forums, Robert has written for *The SOA Magazine*, been quoted in *CIO Insight*, *Telecommunications*, *Infoworld*, and *Computerworld*, and has co-authored two books including *SOA Governance* (IBM Press 2008) and *Executing SOA* (IBM Press 2008).

Anne Thomas Manes

Anne Thomas Manes is the Vice President and Research Director for Burton Group Application Platform Strategies. Her expertise includes SOA, web services, XML, governance, Java, application servers, super platforms, and application security.

Prior to joining Burton Group, Anne was the Chief Technology Officer at Systinet, an SOA governance vendor (now part of HP) and Director of Market Innovation in Sun Microsystems's software group. With 28 years of experience, Anne was named one of the 50 most powerful people in networking 2002 by Network World and among the "Power 100 IT Leaders," by Enterprise Systems Journal.

Anne has authored *Web Services: A Manager's Guide* (Addison-Wesley, 2003) and contributed the foreword for the new book *Next Generation SOA* (Prentice Hall, 2011). Anne has also participated in Web services standards development efforts at the W3C, OASIS, WS-I, and JCP.

Robert Schneider

Robert Schneider is a Partner at WiseClouds, LLC. WiseClouds offers vendor-neutral, unbiased consulting and training services that help customers understand and manage cloud computing business concerns, select the right mixture of enabling technologies, and identify and deploy the ideal configuration required.

Robert has provided database optimization, distributed computing, and other technical expertise to a wide variety of enterprises in the financial, technology, and public sectors. Clients have included Amazon, JP Morgan Chase & Co, VISA, HP, S.W.I.F.T., and numerous governments such as the United States, Brazil, Malaysia, Mexico, Australia, and the United Kingdom.

Robert has written six books and numerous articles on database technology and other complex topics such as cloud computing, and SOA. Robert is a frequent organizer and presenter at technology industry events, worldwide.

Leo Shuster

Leo Shuster is a seasoned IT professional. He has directed Enterprise Architecture and SOA strategy and execution for a number of organizations including Nationwide Insurance, National City Corporation, Ohio Savings Bank, and Progressive Insurance.

Leo holds an MS in Computer Science and Engineering from Case Western Reserve University and an MBA from Cleveland State University. Thus far, in his 15 year IT career, Leo has held a variety of roles including Director, Manager, Team Lead, Project Manager, Architect, and Developer. Leo has presented on Enterprise Architecture, SOA, and related topics for groups of all sizes at a variety of industry events and conferences. He is passionate about technology and regularly blogs about advanced software architecture issues at leoshuster.blogspot.com.

Andre Tost

Andre Tost works as a Senior Technical Staff Member in the IBM Software Group where he assists IBM's customers in establishing service-oriented architectures. His special focus is on Web services, Enterprise Service Bus technology, and SOA governance.

Before his current assignment, Andre spent ten years in various partner enablement, development, and architectural roles in IBM software development. Andre has spoken at industry conferences worldwide on topics related to SOA and is a frequent publisher of articles and papers. He is also a co-author of several books on Web services and related technologies including *Web Service Contract Design and Versioning for SOA* (Prentice Hall 2008). Originally from Germany, he now lives and works in Rochester, Minnesota.

Chris Venable

Chris Venable is an architect and member of the SOA Center of Competency at Wal-Mart Stores, Inc. He has 16 years of experience in the IT industry with the past nine focused on SOA, data integration, and other modern software engineering practices. Current areas of interest include business architecture, event processing, variation analysis, and conceptual modeling.

This page intentionally left blank

About the Contributors

Benjamin Carlyle

Benjamin is a long-time Australian employee of Invensys Rail Group and has been involved in projects worldwide as a founding developer of the “SystematICS” product line, as a software architect and systems engineer. Benjamin has been working with REST and related technologies since 2004 through his blog (soundadvice.id.au/blog), his day job, and various other forums.

Benjamin’s work has been referenced in books on RESTful Web services and on micro-formats. He has presented at the SOA Symposium, was a member of the program committee for the first and second international workshop on RESTful Design, and is co-author of *SOA with REST* (Prentice Hall, 2011). He is credited with helping inspire the RESTlet framework for Java and coined the term “REST Triangle” to describe the structure of a REST uniform contract. He has a deep understanding of both the theory and practice of REST and related styles, as well as broader software and systems architecture topics.

Robert Moores

Robert Moores lives and works in Leeds, England, where he is currently Head of Information & Media Services at Leeds Metropolitan University. Robert has worked in IT for more than 25 years, starting his career as a COBOL programmer. Robert’s career has encompassed higher education, government, health, and insurance sectors and he has worked in the UK, Australia, and the USA as a Developer, Business Analyst, Project Manager and most recently as an Operational Manager. You can read more writings from Robert at robmoores.wordpress.com.

Filippos Santas

Filippos is an IT Architect for Credit Suisse Private Banking in Switzerland and an SOASchool.com Certified SOA Trainer, Analyst, Architect, Consultant, and Security Specialist; additionally, he is certified by IBM on the Rational Unified Process V7.0. In the last 10 years, Filippos has been the lead for the analysis and architecture of a number of successful SOA projects and for transforming legacy architectures to service-oriented architectures, primarily in the financial sector. His achievements include the conceptual and physical design of the international insurance claims service network that connects insurance policies, customer companies, and state authorities in 44 countries across the globe, combining legislation, business processes, business rules, business objects, knowledge bases, and different platforms.

Filippos's recent work with Credit Suisse has been as an SOA Architect, combining the profitable and established functionality of legacy systems with centralized business process services, decentralized business rule services, domain specific BOMs and entity services, end-to-end traceability, and service composability at every level.

About the Foreword Contributors

Massimo Pezzini

Massimo Pezzini is a vice president and distinguished analyst in Gartner Research. His research focus is currently on application platforms, ultra high-end transaction processing technology, composite applications, service-oriented and event-driven architecture infrastructure and best practices, and application integration including integration appliances and mobile middleware. Pezzini has decades of experience in distributed computing, middleware technology, and software architectures. Throughout his career, he has had responsibilities in software development, project management, pre-sales support, consulting, and product marketing. Prior to joining Gartner, Mr. Pezzini was managing director of the Internet Business Unit of Infostrada, an Italian telecom provider. Before that, he held various positions in the Olivetti group.

Roberto Medrano

A recognized executive in the information technology fields of SOA, Internet security, governance, and compliance, Medrano has extensive experience with both start-ups and large companies, having been involved at the beginning of four IT industries: EDA, Open Systems, Computer Security, and now SOA.

Medrano holds an MBA from UCLA, a MSEE from MIT, BSEE from USC and prior to joining SOA Software, he was CEO of PoliVec, a leader in security policy. Before joining PoliVec, he was one of the top 100 Sr. Executives at Hewlett-Packard. At Hewlett-Packard (HP) he served as the General Manager of the E-Services and Internet Security Divisions. Medrano has held executive positions at Finjan, Avnet Inc, and Sun Microsystems. Medrano participated in President Clinton's White House Security Summit and

has been an active member on National Cyber Security Summits and the White House National Strategy to Secure Cyberspace. Medrano has been selected as one of “The 100 most influential Hispanics in US,” “The 100 most influential Latinos in Silicon Valley” “Top 100 most influential Hispanics in Information Technology” and is co-founder and CEO for Hispanic-Net, a non-profit organization.

Index

A

acquisition strategies for governance technology, 444-447
active governance tasks, 429
activities in RUP, 620. *See also* vitality activities
administrative governance technology, 429
Adoption Impact Analysis process, 176-178
adoption planning
 case study, 182-186
 people for, 179-182
 precepts for, 169-173
 processes for, 173-178
Adoption Risk Assessment process, 178
Agnostic Capability design pattern, 491
Agnostic Context design pattern, 492
agnostic logic, defined, 39-40
Agnostic Sub-Controller design pattern, 493
analysis stages in MSOAM, mapping to RUP disciplines, 626-627. *See also* Service Inventory Analysis lifecycle; Service-Oriented Analysis stage
Annotated SOA Manifesto, 34, 53, 578-590

approve activity (vitality activities), 423
artifacts in RUP, 620
assess activity (vitality activities), 422
assessing governance technology, 448-449
Asynchronous Queuing design pattern, 494
Atomic Service Transaction design pattern, 495
attachments (SOAP), 483
attacks, types of, 162
authentication. *See* Brokered Authentication design pattern; Direct Authentication design pattern
automated governance tasks, 427

B

backwards compatibility, 596-599
balanced scope, 51, 53-55. *See also* pillars of service-orientation
best practices
 assessing governance technology, 448-449
 in SOA governance program implementation, 146-150
books related to this book, 5-6
breadth roles in RUP, 623-624

Brokered Authentication design pattern, 162, 496
Business Aligned maturity level, 58
Business Analyst role, 113, 397-399
business changes versus technology
 changes in vitality triggers, 415-416
business dictionaries, 375-376
Business Driven maturity level, 58
business heat map, 195
business information, assigning
 value to, 409
Business Policy Standards precept, 382-384
Business Requirements Prioritization
 process, 195-197
business shifts, as vitality triggers, 417

C

CA (certificate authority), 161
Canonical Data Model
 design pattern, 224
Canonical Expression
 design pattern, 206, 225, 497
Canonical Protocol
 design pattern, 225, 498
Canonical Resources
 design pattern, 499
Canonical Schema Bus
 compound pattern, 501
Canonical Schema
 design pattern, 224, 500
Canonical Versioning
 design pattern, 610, 502
Canonical XML, 161
Capability Composition
 design pattern, 207, 503
capability granularity, defined, 45
capability profile structure, 118-119
Capability Recomposition
 design pattern, 504
capitalization usage, 14

case studies (Raysmoore Corporation)
 background, 18-20
 conclusion, 454-455
 precepts (governance controls),
 167-168
Service Deployment and
 Maintenance, 312-313
Service Development, 276
Service Discovery, 350-351
Service Inventory Analysis, 201-205
Service Logic Design, 265-266
Service-Oriented Analysis, 217-220
Service Testing, 294-297
Service Usage and Monitoring,
 333-334
SOA adoption planning, 182-186
central funding model
 in platform funding, 61, 64-66
 in service funding, 69, 71-72
centralized domain SOA Governance
 Program Office, 134
centralized enterprise SOA Governance
 Program Office, 133
Centralized Service Registry
 precept, 335-337
certificate authority (CA), 161
Certified Cloud Computing, 15-16
cloud, defined, 36
cloud-based security groups, 161
Cloud Burst Threshold, 318
cloud computing, defined, 35
Cloud Computing Governance
 Specialist role, 114
Cloud Computing Security Specialist
 role, 114
cloud consumers, defined, 38
cloud delivery models, list of, 38
cloud deployment models, list of, 37-38
cloud governance tasks, 428
Cloud Integration Testing Standards
 precept, 283-284

- cloud providers, defined**, 38
- Cloud Resource Administrator role**, 100-102
 - reference diagram, 462
 - for Service Deployment and Maintenance, 305-306
 - in Service Testing, 288
 - in Service Usage and Monitoring, 328
 - in Service Versioning and Retirement, 363
- Cloud Service Owner role**, 98-99
- cloud services**, defined, 36
- cloud vendor leasing acquisition strategy**
 - for governance technology, 447
- coarse-grained constraints**, 595
- code examples**
 - backwards compatibility
 - for methods, 598
 - for REST services, 597
 - for Web services, 596-597
 - in XML Schemas, 598-599
 - complexType construct containing fine and coarse-grained constraints, 595
 - default value of minOccurs attribute, 602
 - forwards compatibility with wildcards, 601
 - incrementing minOccurs attribute value, 604
- communicate activity (vitality activities)**, 423
- Communications Quality Review process**, 391
- community cloud deployment model**, 37
- compatibility, versioning and**, 596-605
 - backwards compatibility, 596-599
 - compatible changes, 602-604
 - forwards compatibility, 599-602
- incompatible changes, 604-605
- REST services compatibility, 605-608
- compatibility guarantee**, 609
- Compatible Change design pattern**, 353, 505
- compatible changes**, 602-604
- Compensating Service Transaction design pattern**, 506
- complexity of versioning strategies**, 615
- compliance metrics**, 166, 419
- components, services as**, 32
- Composition Autonomy design pattern**, 507
- composition controller capabilities**, design characteristics, 487
- composition member capabilities**, design characteristics, 486
- compound patterns**
 - Canonical Schema Bus, 501
 - Enterprise Service Bus, 523
 - Federated Endpoint Layer, 527
 - Official Endpoint, 539
 - Orchestration, 540
 - Service Broker, 553
 - Three-Layer Inventory, 570
- computing professionals, Certified Cloud Computing**, 15-16
- Concurrent Contracts design pattern**, 225, 508
- configuration management tools**, 443
- constraint granularity**, 45, 595
- content sharing and publishing tools**, 442-443
- Contract Centralization design pattern**, 225, 509
- Contract Denormalization design pattern**, 510
- “contract first” approach**, 476
- cost metrics**, 164

cost of capital, 164
Cross-Domain Utility Layer design pattern, 511
Custom Development Technology Standards precept, 268-270
custom SOA governance solutions, 443-444

D

data, defined, 372
Data Architect role, 113, 399
Data Confidentiality design pattern, 162, 512
Data Format Transformation design pattern, 513
data granularity, defined, 45
Data Model Transformation design pattern, 514
Data Origin Authentication design pattern, 162, 515
Data Quality Review process, 389-391
deactivation. *See Service Versioning and Retirement stage*
Decomposed Capability design pattern, 516
Decoupled Contract design pattern, 225, 517
Decryption Transform for XML Signature, 161
Define Enterprise Business Models in Service Inventory Analysis lifecycle, 408-409, 626
Define Technology Architecture process, 626
Definition of the Service Inventory Blueprint process, 627
delivery models, 38
deployment. *See cloud deployment models; Service Deployment and Maintenance stage*

depth roles in RUP, 623-624
design patterns, 13-14

- Agnostic Capability**, 491
- Agnostic Context**, 492
- Agnostic Sub-Controller**, 493
- Asynchronous Queuing**, 494
- Atomic Service Transaction**, 495
- Brokered Authentication**, 162, 496
- Canonical Data Model**, 224
- Canonical Expression**, 206, 225, 497
- Canonical Protocol**, 225, 498
- Canonical Resources**, 499
- Canonical Schema**, 224, 500
- Canonical Versioning**, 502, 610
- Capability Composition**, 207, 503
- Capability Recomposition**, 504
- Compatible Change**, 353, 505
- Compensating Service Transaction**, 506
- Composition Autonomy**, 507
- Concurrent Contracts**, 508
- Contract Centralization**, 225, 509
- Contract Denormalization**, 510
- Cross-Domain Utility Layer**, 511
- Data Confidentiality**, 162, 512
- Data Format Transformation**, 513
- Data Model Transformation**, 514
- Data Origin Authentication**, 162, 515
- Decomposed Capability**, 516
- Decoupled Contract**, 225, 517
- defined**, 46-47
- Direct Authentication**, 162, 518
- Distributed Capability**, 519
- Domain Inventory**, 54, 193, 520
- Dual Protocols**, 225, 521
- Enterprise Inventory**, 193, 522
- Entity Abstraction**, 54, 524
- Event-Driven Messaging**, 525
- Exception Shielding**, 162, 526
- File Gateway**, 528
- Functional Decomposition**, 529

- Intermediate Routing, 530
 - Inventory Endpoint, 531
 - Legacy Wrapper, 249, 532
 - Logic Centralization, 533
 - Message Screening, 162, 534
 - Messaging Metadata, 378, 535
 - Metadata Centralization, 234, 536
 - Multi-Channel Endpoint, 537
 - Non-Agnostic Context, 538
 - Partial State Deferral, 541
 - Partial Validation, 542
 - Policy Centralization, 388, 543
 - Process Abstraction, 54, 544
 - Process Centralization, 545
 - Protocol Bridging, 546
 - Proxy Capability, 353, 547
 - Redundant Implementation, 548
 - Reliable Messaging, 549
 - Rules Centralization, 550
 - Schema Centralization, , 224, 551
 - Service Agent, 552
 - Service Callback, 554
 - Service Data Replication, 555
 - Service Decomposition, 353, 556
 - Service Encapsulation, 557
 - Service Façade, 249, 558
 - Service Grid, 559
 - Service Instance Routing, 560
 - Service Layers, 54, 561
 - Service Messaging, 562
 - Service Normalization, 207,
 - 344, 563
 - Service Perimeter Guard, 162, 564
 - Service Refactoring, 353, 565
 - State Messaging, 566
 - State Repository, 567
 - Stateful Services, 568
 - Termination Notification, 356, 569
 - Trusted Subsystem, 162, 571
 - UI Mediator, 572
 - Utility Abstraction, 54, 573
 - Validation Abstraction, , 225, 574
 - Version Identification, 353, 575, 608
- design principles, 13-14**
- list of, 27
 - Service Abstraction, 228, 225,
 - 374, 478
 - Service Autonomy, 481
 - Service Composability, 88, 486-487
 - Service Discoverability, 91, 225, 228,
 - 234, 239, 335, 391, 484-485
 - Service Loose Coupling, 225, 226,
 - 228, 477
 - Service Reusability, 479-480
 - Service Statelessness, 482-483
 - Standardized Service Contract, 87,
 - 225, 228, 237, 475-476
- design stages in MSOAM, mapping to RUP disciplines, 626-627**
- design-time governance tasks, 428**
- digital certificates, 161**
- digital signatures, 160**
- Direct Authentication design pattern, 162, 518**
- disciplines, 51-52. *See also* pillars of service-orientation**
- in RUP, 620
 - mapping to MSOAM analysis and design stages, 626-627*
 - mapping to MSOAM service delivery project stages, 625-626*
- Distributed Capability design pattern, 519**
- Domain Business Dictionary precept, 375-376**
- Domain Inventory design pattern, 54, 193, 520**
- Domain Ontology precept, 380-382**
- domain service inventory, defined, 41**
- domains, assessing, 137-139**
- Dual Protocols design pattern, 225, 521**

E

education, 51-52. *See also* pillars of service-orientation

Educator, 112

embedded policy logic, 374

encryption, 160

enforcement governance

technology, 430

enterprise, assessing, 137-139

Enterprise Architect role, 106

 reference diagram, 467

 in Service Deployment and Maintenance, 308

 in Service Development, 274

 in Service Inventory Analysis, 199

 in Service Logic Design, 261

 in Service-Oriented Analysis, 215

 in Service-Oriented Design, 242

 in Service Testing, 289

 in Service Usage and Monitoring, 325-326

 in SOA adoption planning, 179-180

Enterprise Business Dictionary precept, 375-376

enterprise business models, establishing, 408-409

Enterprise Design Standards Custodian role, 107-108

 reference diagram, 468

 in Service Development, 273-274

 in Service Inventory Analysis, 198-199

 in Service Logic Design, 260

 in Service-Oriented Analysis, 214

 in Service-Oriented Design, 241-242

 in Service Versioning and Retirement, 360-361

Enterprise Inventory design pattern, 193, 522

Enterprise Ontology precept, 380-382

enterprise roles in MSOAM, 624-626

Enterprise Service Bus compound pattern, 523

Enterprise Unified Process (EUP), 628

Entity Abstraction design pattern, 54, 524

entity services, defined, 39

entry fees in usage funding model, 66

EUP (Enterprise Unified Process), 628

Event-Driven Messaging design pattern, 525

Exception Shielding design pattern, 162, 526

F

federated domain SOA Governance Program Offices, 135

Federated Endpoint Layer compound pattern, 527

fees in usage funding model, 66

File Gateway design pattern, 528

fine-grained constraints, 595

flexible versioning strategy, 611-613

forwards compatibility, 599-602

Functional Decomposition design pattern, 529

functional metadata, 378

functional tests, 278

funding models, 60-77

 platform funding, 60-69

 service funding, 69-74

 Standardized Funding Model

 precept, 172-173

G

gaps in RUP and service-orientation, 628

glossary Web site, 5, 15-16, 632

governance

 defined, 122-123

 management and, 124-126

- MDM (master data management)
and, 409
methodology and, 124-126
scope of, 123-126
selecting style of, 126-127
SOA and, 130
SOA Governance Program Office (SGPO), 131-136
vitality. *See* vitality
- governance controls**, 127-129
metrics, 129, 146, 164-165
people. *See* organizational roles
precepts. *See* precepts (governance controls)
processes. *See* processes (governance controls)
- governance impact of versioning strategies**, 614
- governance roles in MSOAM**, 624-626
- governance systems**, defined, 426-427
- governance task types**, 427-429
- governance technology**
acquisition strategies, 444-447
assessing, 448-449
categories of, 429-431
product types
configuration management tools, 443
content sharing and publishing tools, 442-443
custom SOA governance solutions, 443-444
policy systems, 437-439
quality assurance tools, 439-441
repositories, 433-435
service agents, 435-437
service registries, 431-433
SOA management suites, 441-442
technical editors and graphic tools, 442
- granularity, service-related granularity**,
defined, 44-45
- guidelines**, defined, 128
- H**
- hardened virtual server images**, 161
- hardware accelerators**, 483
- hashing**, 160
- human-readable policies**, 373
- hybrid cloud deployment model**, 38
- hybrid funding model in service funding**, 69, 72-74
- I**
- IaaS (Infrastructure-as-a-Service delivery model**, 38
- identify activity (vitality activities)**, 421
- identity and access management (IAM)**, 160
- implementation requirements, service contracts**, 475
- incompatible changes**, 604-605
- independent domain SOA Governance Program Offices**, 136
- industry shifts, as vitality triggers**, 417-418
- information**
business information, assigning value to, 409
defined, 372
- Information Alignment Audit process**, 393-395
- Infrastructure-as-a-Service (IaaS delivery model**, 38
- integration costs**, 164
- integration tests**, 279
- Intermediate Routing design pattern**, 530
- Inventory Endpoint design pattern**, 531

IT Manager role, 115
IT resources, defined, 35-36
IT roles, 112-115

J-K-L

jurisdiction models in SOA Governance Program Office (SGPO), 133-136
knowledge, defined, 372
leasing from cloud vendor acquisition strategy for governance technology, 447
Legacy Wrapper design pattern, 249, 532
Legal Data Audit process, 257-258
lifecycle stages. *See service project lifecycle stages*
locked-in costs, 164
Logic Centralization design pattern, 533
logical domain precepts, 159
loose versioning strategy, 611, 613-614

M

Mainstream SOA Methodology.
See **MSOAM (Mainstream SOA Methodology)
maintenance, 298
management
 governance and, 124-126
 methodology and, 125-126
manual governance tasks, 427
mapping diagrams, 12
master data management (MDM),
 governance and, 409
maturity levels in SOA planning, 56-59
message-layer security, 160
Message Screening design pattern,
 162, 534
Messaging Metadata design pattern,
 378, 535**

metadata, 377-380
Metadata Centralization design pattern,
 234, 536
methodology

governance and, 124-126
 management and, 125-126
metrics (governance controls), 127,
 129, 146
 cost metrics, 164
 standards-related precept
 metrics, 165
 threshold metrics, 165
 as vitality triggers, 418-419
milestone triggers, 420
monitoring governance technology, 429
MSOAM (Mainstream SOA Methodology)
 analysis and design stages, mapping
 to RUP disciplines, 626-627
 roles in
enterprise and governance roles,
 624-626
mapping to RUP roles, 623-624
 service delivery project stages,
 mapping to RUP disciplines,
 625-626

Multi-Channel Endpoint design pattern, 537
multiple vendor acquisition strategy for governance technology, 445-446

N

naming standards, Service and Capability Candidate Naming Standards precept, 206
Non-Agnostic Context design pattern, 538
non-agnostic logic, defined, 39-40
notification service for this book series,
 16, 632

O

objectives, defined, 128
Official Endpoint compound pattern, 539
on-going costs, 164
on-premise, defined, 37
on-premise governance tasks, 428
ontologies, 380-382
open source acquisition strategy for governance technology, 446-447
Operational Policy Standards precept, 384-386
Orchestration compound pattern, 540
Organizational Governance Maturity Assessment process, 173-175
organizational maturity, levels of, 56-59
Organizational Maturity Criteria Definition precept, 171
organizational roles, 92-115, 127-128, 156
 Business Analysts, 397-399
 Cloud Resource Administrator, 100-102, 462
 Cloud Service Owner, 98-99
 Data Architects, 399
 Educator, 112
 Enterprise Architect, 106, 467
 Enterprise Design Standards Custodian, 107-108, 468
 IT roles, 112-115
 planning and building SOA governance programs, 143
 Policy Custodian, 104, 401, 464
 Schema Custodian, 102-103, 399-400, 463
 Service Administrator, 100, 461
 Service Analyst, 96, 458
 Service Architect, 96, 459
 Service Custodian, 98, 460
 for Service Deployment and Maintenance, 304-311

Service Developer, 97, 460
for Service Development, 272-275
for Service Discovery, 345-348
for Service Inventory Analysis, 197-200
for Service Logic Design, 259-264
for Service-Oriented Analysis, 212-217
for Service-Oriented Design, 236-246
Service Registry Custodian, 105, 402-403, 465
for Service Testing, 287-293
for Service Usage and Monitoring, 325-332
for Service Versioning and Retirement, 360-366
for SOA adoption planning, 179-182
SOA Governance Specialist, 111, 406-407, 471-472
SOA Quality Assurance Specialist, 109, 405-406, 469
SOA Security Specialist, 110, 470
Technical Communications Specialist, 105, 403, 466
organizational shifts as vitality triggers, 419-420

P

PaaS (Platform-as-a-Service) delivery model, 38
Partial State Deferral design pattern, 541
Partial Validation design pattern, 542
passive governance tasks, 428
patterns. *See design patterns*
people (governance controls). *See organizational roles*
performance, state management and, 483
performance metrics, 166, 419
performance tests, 279

- periodic vitality triggers**, 420
- per use fees in usage funding model**, 66
- pillars of service-orientation**, 51-55
 - balanced scope, 53-55
 - discipline, 52
 - education, 52
 - mapping to RUP principles, 620-622
 - teamwork, 52
- PKI (Public Key Infrastructure)**, 161
- planning. *See also* SOA planning**
 - SOA adoptions
 - case study*, 182-186
 - people for*, 179-182
 - precepts for*, 169-173
 - processes for*, 173-178
 - SOA governance programs, 139-146
- Platform-as-a-Service (PaaS) delivery model**, 38
- platform funding models**, 60-69
 - central funding model, 64-66
 - project funding model, 61
 - usage funding model, 66-69
- policies**
 - defined, 128
 - explained, 373-374
 - WS-Policy assertions, 355
- Policy Centralization design pattern**, 386-388, 543
- Policy Conflict Audit process**, 395-397
- Policy Custodian role**, 104, 401
 - reference diagram, 464
 - in Service Deployment and Maintenance, 311
 - in Service-Oriented Design, 238
 - in Service Versioning and Retirement, 364
- policy systems**, 437-439
- policy tests**, 278
- precepts (governance controls)**, 127-128, 156
 - Business Policy Standards, 382-384
 - case study, 167-168
 - Enterprise Business Dictionary/Domain Business Dictionary, 375-376
 - Enterprise Ontology/Domain Ontology, 380-382
 - logical domain precepts, 159
 - Operational Policy Standards, 384-386
- planning and building SOA**
 - governance programs, 139-141
 - Policy Centralization, 386-388
 - security control precepts, 160-163
 - for Service Deployment and Maintenance, 298-300
 - for Service Development, 267-270
 - for Service Discovery, 335-337
 - service information precepts, 158
 - for Service Inventory Analysis, 193-195
 - for Service Logic Design, 249-253
 - Service Metadata Standards, 377-380
 - for Service-Oriented Analysis, 206-210
 - for Service-Oriented Design, 223-231
 - service policy precepts, 158
 - service profile standards, 157
 - for Service Testing, 279-286
 - for Service Usage and Monitoring, 317-322
 - for Service Versioning and Retirement, 352-356
- for SOA adoption planning**, 169-173
- SOA governance technology standards**, 163

- Preferred Adoption Scope Definition precept**, 169-170
- Prentice Hall Service-Oriented Computing Series from Thomas Erl**, 632
- principle profiles**
- Service Abstraction, 478
 - Service Autonomy, 481
 - Service Composability, 486-487
 - Service Discoverability, 484-485
 - Service Loose Coupling, 477
 - Service Reusability, 479-480
 - Service Stateless, 482-483
 - Standardized Service Contract, 475-476
- principles of RUP (Rational Unified Processing), mapping to pillars of service-orientation**, 620-622
- private cloud deployment model**, 38
- private service registries**, 432
- Process Abstraction design pattern**, 54, 544
- Process Centralization design pattern**, 545
- processes (governance controls)**, 127, 129, 156
 - Communications Quality Review, 391
 - Data Quality Review, 389-391
 - Information Alignment Audit, 393-395
 - planning and building SOA governance programs, 141-142
 - Policy Conflict Audit, 395-397
 - for Service Deployment and Maintenance, 301-304
 - for Service Discovery, 337-344
 - for Service Inventory Analysis, 195-197
 - for Service Logic Design, 253-258
 - for Service-Oriented Analysis, 210-211
- for Service-Oriented Design**, 231-235
- for Service Testing**, 286
- for Service Usage and Monitoring**, 323-324
- for Service Versioning and Retirement**, 357-360
- for SOA adoption planning**, 173-178
- Production Deployment and Maintenance Standards precept**, 298-300
- profiles**. *See service profiles*
- programming logic metadata**, 378
- project funding model**
- in platform funding, 61
 - in service funding, 69-70
- project lifecycle stages**. *See service project lifecycle stages*
- Protocol Bridging design pattern**, 546
- Proxy Capability design pattern**, 353, 547
- public cloud deployment model**, 37
- Public Key Infrastructure (PKI)**, 161
- Q–R**
- quality assurance, SOA Quality Assurance Specialist role**, 109
- quality assurance tools**, 439-441
- quality of service metadata**, 378
- Rational Unified Process**. *See RUP (Rational Unified Process)*
- Raysmoore Corporation case study**.
See case studies (Raysmoore Corporation)
- recommended reading**, 5-6, 14-16, 47-48, 628
- Redundant Implementation design pattern**, 548
- refresh activity (vitality activities)**, 422-423

regression tests, 278
Reliable Messaging design pattern, 549
 reporting governance technology, 430
 repositories, 433-435
 resources, 35-36
responsibilities. *See organizational roles*
REST services
 compatibility considerations, 605-608
 defined, 34
 versioning, 594-595
 backwards compatibility, 597-599
 forwards compatibility, 600
 strategy considerations, 615-616
retirement. *See Service Versioning and Retirement stage*
RFPs (requests for proposal), creating, 449
roles. *See also organizational roles*
 in MSOAM, enterprise and governance roles, 624-626
 in RUP, 619, 623-624
Rules Centralization design pattern, 550
 runtime governance tasks, 428
Runtime Service Usage Thresholds precept, 317-319
RUP (Rational Unified Process), 618
 breadth and depth roles, 623-624
 compatibility with SOA, 618-619, 628
 content elements of, 619-620
 disciplines in
 mapping to MSOAM analysis and design stages, 626-627
 mapping to MSOAM service delivery project stages, 625-626
 principles of, mapping to pillars of service-orientation, 620-622

S

SaaS (Software-as-a-Service) delivery model, 38
SAML (Security Assertion Markup Language), 161
scalability, 482
Schema Centralization design pattern, 224, 551
Schema Custodian role, 102-103, 399-400
 reference diagram, 463
 in Service Deployment and Maintenance, 311
 in Service-Oriented Design, 237-238
 in Service Versioning and Retirement, 364
Schema Design Standards precept, 223-225
scope
 of governance, 123-126
 Service Inventory Scope Definition precept, 193-195
Security Assertion Markup Language (SAML), 161
security attacks, types of, 162
security control precepts, 160-163
security policies, 160
security sessions, 160
security tests, 278
security token actions, 160
selecting style of governance, 126-127
Service Abstraction design principle, 27, 225, 228, 374, 478
Service Access Control process, 253
Service Administrator role, 100
 reference diagram, 461
 in Service Deployment and Maintenance, 304-305
 in Service Testing, 287

- in Service Usage and Monitoring, 327-328
- in Service Versioning and Retirement, 362
- Service Agent design pattern, 552**
- service agents, 435-437**
- Service Aggressive maturity level, 59**
- Service Analyst role, 96**
 - reference diagram, 458
 - in Service Inventory Analysis, 197
 - in Service-Oriented Analysis, 212-213
- Service and Capability Candidate Naming Standards precept, 206**
- Service Architect role, 96**
 - reference diagram, 459
 - in Service Logic Design, 259-260
 - in Service-Oriented Analysis, 213
 - in Service-Oriented Design, 236-237
 - in Service Usage and Monitoring, 326-327
- Service Autonomy design principle, 27, 481**
- Service Aware maturity level, 57**
- Service Billing Threshold, 318**
- Service Broker compound pattern, 553**
- Service Callback design pattern, 554**
- Service Candidate Review process, 210-211**
- Service Candidate Versioning Standards precept, 209**
- service candidates, defined, 42**
- Service Capable maturity level, 57**
- service catalogs, service profiles and, 119. *See also* service portfolio**
- Service Certification Review process, 301-302**
- Service Composability design principle, 27, 88, 486-487**
- Service Composition Membership Threshold, 317**
- service compositions, defined, 40-41**
- Service Contract Design Review process, 231-232**
- Service Contract Design Standards precept, 225-227**
- Service Contract Registration process, 234-235**
- service contracts, 486. *See also* Service-Oriented Design stage**
 - defined, 43-44
 - versioning. *See* versioning
- Service Custodian role, 98**
 - reference diagram, 460
 - in Service Deployment and Maintenance, 307
 - in Service Discovery, 345-346
 - in Service Usage and Monitoring, 329
- Service Data Replication design pattern, 555**
- Service Data Throughput Threshold, 318**
- Service Decomposition design pattern, 353, 556**
- service delivery project stages**
 - in MSOAM, mapping to RUP disciplines, 625-626
- Service Deployment and Maintenance stage, 298**
 - case study, 312-313
 - people for, 304-311
 - precepts for, 298-300
 - processes for, 301-304
 - in service project lifecycle stages, 89
- Service Developer role, 97**
 - reference diagram, 460
 - in Service Development, 272

- Service Development stage**
case study, 276
people for, 272-275
precepts for, 267-270
in service project lifecycle stages, 87
- Service Discoverability design principle**, 27, 91, 225, 228, 234, 239, 335, 391, 484-485
- Service Discovery stage**, 340-341
case study, 350-351
people for, 345-348
precepts for, 335-337
processes for, 337-344
in service project lifecycle stages, 90-91
- Service Elasticity Threshold**, 318
- Service Encapsulation design pattern**, 557
- Service Exception Threshold**, 318
- Service Façade design pattern**, 249, 558
- service funding**, 60, 69-74
central funding model, 71-72
hybrid funding model, 72-74
project funding model, 69-70
usage funding model, 74
- service granularity**, defined, 44
- Service Grid design pattern**, 559
- Service Ineffectual maturity level**, 58
- Service Information Governance Council**, establishing, 408
- service information precepts**, 158
- Service Instance Routing design pattern**, 560
- Service Instance Threshold**, 317
- service inventory**, defined, 41
- Service Inventory Analysis lifecycle**, 83, 626
case study, 201-205
Define Enterprise Business Models step, 408-409
- iterative cycles in, 192
people in, 197-200
precepts for, 193-195
processes for, 195-197
in service project lifecycle stages, 82-83
time allotted to, 189-190
- service inventory blueprints**, defined, 41
- service inventory funding models**.
See platform funding models
- Service Inventory Scope Definition precept**, 193-195
- Service Layers design pattern**, 54, 561
- Service Logic Design Review precept**, 255-257
- Service Logic Design stage**
case study, 265-266
people for, 259-264
precepts for, 249-253
processes for, 253-258
in service project lifecycle stages, 87
- Service Logic Design Standards precept**, 249-251
- Service Logic Programming Standards precept**, 267-268
- Service Loose Coupling design principle**, 27, 225-226, 228, 477
- service maintenance**, service versioning versus, 298
- Service Maintenance Review process**, 303-304
- Service Messaging design pattern**, 562
- Service Metadata Standards precept**, 377-380
- service modeling process**, 84-85. *See also* Service-Oriented Analysis stage
- service models**, defined, 38-40
- Service Monitoring Footprint Threshold**, 318

- Service Neutral maturity level**, 57
- Service Normalization design pattern**, 207-209, 344, 563
- service-orientation**
defined, 26-27
pillars of, 51-55
mapping to RUP principles, 620-622
RUP and, gaps in, 628
- Service-Orientation Architecture**
Design Standards precept, 252-253
- Service-Orientation Contract Design Standards precept**, 228
- Service-Oriented Analysis stage**
case study, 217-220
people in, 212-217
precepts for, 206-210
processes for, 210-211
in service project lifecycle stages, 84-85
time allotted to, 189-190
- Service-Oriented Architecture: Concepts, Technology, and Design**, 5, 80
- service-oriented architecture (SOA)**, defined, 29
- service-oriented computing**, defined, 25-26
- Service-Oriented Design stage**
people for, 236-246
precepts for, 223-231
processes for, 231-235
in service project lifecycle stages, 85-86
- Service Perimeter Guard design pattern**, 162, 564
- service policy precepts**, 158
- service portfolio**, defined, 41-42
- service profiles**, 115-120
capability profile structure, 118-119
service catalogs and, 119
- service registries and, 119
structure of, 117
- service profile standards**, 157
- service project lifecycle stages**, 81-91
Service Deployment and Maintenance, 89
Service Development, 87
Service Discovery, 90-91
Service Inventory Analysis, 82-83
Service Logic Design, 87
Service-Oriented Analysis, 84-85
Service-Oriented Design, 85-86
Service Testing, 88-89
Service Usage and Monitoring, 90
Service Versioning and Retirement, 91
SOA Adoption Planning, 82
- Service Refactoring design pattern**, 353, 565
- service registries**, 431-433. *See also Service Discovery stage*
Centralized Service Registry
precept, 335-337
service profiles and, 119
Service Registry Access Control process, 337-339
Service Registry Record Review process, 339
- Service Registry Access Control process**, 337-339
- Service Registry Custodian role**, 105, 402-403
reference diagram, 465
in Service Discovery, 346-347
- Service Registry Record Review process**, 339
- service-related granularity**, defined, 44-45
- Service Retirement Notification precept**, 356

- Service Retirement process**, 359-360
- Service Reusability design principle**, 27, 479-480, 486
- Service Stateless design principle**, 27, 482-483
- Service Testing stage**
- case study, 294-297
 - people for, 287-293
 - precepts for, 279-286
 - processes for, 286
 - in service project lifecycle stages, 88-89
 - types of tests, 278
- Service Testing Standards precept**, 281-283
- Service Test Results Review process**, 286
- Service Usage and Monitoring stage**
- case study, 333-334
 - people for, 325-332
 - precepts for, 317-322
 - processes for, 323-324
 - in service project lifecycle stages, 90
- service versioning, service maintenance versus**, 298
- Service Versioning and Retirement stage**
- people in, 360-366
 - precepts for, 352-356
 - processes for, 357-360
 - in service project lifecycle stages, 91
- Service Versioning process**, 357-358
- Service Versioning Strategy precept**, 352-353
- Service Vitality Review process**, 323-324
- Service Vitality Triggers precept**, 320-322
- services**
- cloud services, defined, 36
 - as components, 32
 - defined, 31-34
 - as REST services, 34
- scalability, 482
- as Web services, 32-33
- SGPO. See SOA Governance Program Office (SGPO)**
- Shared Service Modification Request process**, 343-344
- Shared Service Usage Request process**, 342-343
- single sign-on**, 161
- single vendor acquisition strategy for governance technology**, 444-445
- SLA Template precept**, 229-231
- SLA Versioning Rules precept**, 354-356
- SOA (service-oriented architecture)**
- defined, 29
 - governance and, 130
 - RUP (Rational Unified Process) compatibility with, 618-619
 - scalability, 482
- SOA Adoption Planning stage**
- case study, 182-186
 - people for, 179-182
 - precepts for, 169-173
 - processes for, 173-178
 - in service project lifecycle stages, 82
- SOA Certified Professional (SOACP)**, 15-16
- SOA design patterns. See design patterns**
- SOA Design Patterns**, 5
- SOA governance program**
- implementation, 137-150
 - assessing the enterprise/domain, 137-139
 - best practices, 146-150
 - common pitfalls, 148-150
 - planning and building SOA governance program, 139-146
- SOA Governance Program Office (SGPO)**, 131-132, 155
- jurisdiction models, 133-136

SOA Governance Program Project Plan, 146

SOA Governance Roadmap, 146

SOA Governance Specialist role, 111, 406-407

- reference diagrams, 471-472
- in Service Deployment and Maintenance, 311
- in Service Development, 275
- in Service Discovery, 348
- in Service Inventory Analysis, 199
- in Service Logic Design, 263
- in Service-Oriented Analysis, 216-217
- in Service-Oriented Design, 245-246
- in Service Testing, 292-293
- in Service Usage and Monitoring, 332
- in Service Versioning and Retirement, 365
- in SOA adoption planning, 181

SOA governance technology standards, 163

SOA Governance Tools, 146

SOA governance vitality. *See* **vitality**

SOA Magazine, *The* Web site, 15, 632

SOA management suites, 441-442

SOA Manifesto, 34, 53, 578-590

SOA mapping, RUP and, 628

SOA planning

- funding models, 60-77
 - platform funding*, 60-69
 - service funding*, 69-74
- organizational maturity, levels of, 56-59
- pillars of service-orientation, 51-55

SOA Principles of Service Design, 5, 80

SOA Quality Assurance Specialist role, 109, 405-406

- reference diagram, 469
- in Service Deployment and Maintenance, 309
- in Service Testing, 290-291

SOA Security Specialist role, 110

- reference diagram, 470
- in Service Deployment and Maintenance, 310
- in Service Discovery, 339
- in Service Logic Design, 262
- in Service-Oriented Design, 243
- in Service Testing, 291
- in Service Usage and Monitoring, 331

SOA with REST, 5

SOAP

- attachments, 483
- processors, 483

Software-as-a-Service (SaaS) delivery model, 38

specifications, www.soaspecs.com Web site, 15-16

Standardized Funding Model precept, 172-173

Standardized Service Contract design principle, 27, 87, 225, 228, 237, 475-476

standards, defined, 128

standards compliance tests, 278

standards-related precept metrics, 165

state management

- performance and, 483
- SOAP attachments and, 483

State Messaging design pattern, 566

State Repository design pattern, 567

Stateful Services design pattern, 568

strategic adjustments, as vitality triggers, 416-417

strictness of versioning strategies, 614
 strict versioning strategy, 611-612
 sunk costs, 164
 supplemental fees in usage funding model, 66
 symbols, legend, 12

T

task services, defined, 39
 tasks in RUP, 620
 teamwork, 51, 52. *See also* pillars of service-orientation
Technical Communications Specialist role, 105, 403
 reference diagram, 466
 in Service Discovery, 348
 in Service-Oriented Design, 239-240
technical editors and graphic tools, 442
 technical policies, 373
 technology changes versus business changes, in vitality triggers, 415-416.
See also governance technology
 technology metadata, 378
 technology shifts, as vitality triggers, 418
Termination Notification design pattern, 356, 569
Test Data Usage Guidelines precept, 285
testing. *See* Service Testing stage
Testing Parameter Standards precept, 280
Testing Tool Standards precept, 279-280
Three-Layer Inventory compound pattern, 570
threshold metrics, 165
time triggers, 420
tools, defined, 427
transport-layer security, 160
triggers. *See* vitality triggers
trust brokering, 160
Trusted Subsystem design pattern, 162, 571

U

UI Mediator design pattern, 572
unit tests, 278
up-front costs, 164
usage funding model
 in platform funding, 61, 66-69
 in service funding, 69, 74
usage thresholds, Runtime Service Usage Thresholds precept, 317-319
Utility Abstraction design pattern, 54, 573
utility services, defined, 39

V

Validation Abstraction design pattern, 225, 574
version control systems, 480
Version Identification design pattern, 353, 575, 608
version identifiers, 608-611
versioning. *See also* service versioning
 compatibility and, 596-608
 constraint granularity, 595
 questions concerning, 592-593
 REST services, 594-595
 Service Candidate Versioning Standards precept, 209
 strategies, 611-616
 version identifiers, 608-611
 Web services, 593-594

vitality

defined, 412
 explained, 412
 framework for, 413

vitality activities, 412, 421-424
 approve activity, 423
 assess activity, 422
 communicate activity, 423
 identify activity, 421
 refresh activity, 422-423

vitality triggers, 412, 414-421
business changes versus technology
 changes, 415-416
industry shifts, 417-418
metrics, 418-419
organizational shifts, 419-420
periodic triggers, 420
Service Vitality Review process,
 323-324
Service Vitality Triggers precept,
 320-322
strategic adjustments, 416-417

W

*Web Service Contract Design and
Versioning for SOA, 80*

Web services

 defined, 32-33
 versioning, 593-597

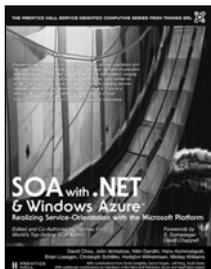
Web sites

 www.cloudschool.com, 15-16, 632
 www.cloudsymposium.com, 633
 www.serviceorientation.com, 633
 www.soabooks.com, 6, 14, 16,
 48, 632
 www.soabooks.com/governance/,
 431
 www.soaglossary.com, 5, 15-16,
 48, 632
 www.soamag.com, 15, 632
 www.soa-manifesto.com, 34, 53, 578
 www.soa-manifesto.org, 34, 53, 578
 www.soapatterns.org, 633
 www.soaprinciples.com, 47, 633
 www.soaschool.com, 15, 632
 www.soaspecs.com, 15, 139, 632
 www.soasympoium.com, 633
 www.whatiscloud.com, 633
 www.whatissoa.com, 47, 633

wisdom, defined, 372
work products in RUP, 620
WS-Policy, 162, 476
WS-Policy assertions, 355
WS-PolicyAttachment, 162
WS-SecureConversation, 161
WS-Security, 161
WS-SecurityPolicy, 161
WS-Trust, 161
WSDL languages, 476

X-Z

XML-Encryption, 161
XML parsers, 483
XML schema languages, 476
XML-Signature, 161



SOA with .NET and Windows Azure: Realizing Service-Orientation with the Microsoft Platform

ISBN 9780131582316

Top Microsoft technology experts team up with Thomas Erl to explore service-oriented computing with Microsoft's latest .NET service technologies and Windows Azure innovations.



Service-Oriented Architecture: Concepts, Technology, and Design

ISBN 9780131858589

Widely regarded as the definitive "how-to" guide for SOA, this best-selling book presents a comprehensive end-to-end tutorial that provides step-by-step instructions for modeling and designing service-oriented solutions from the ground up.



SOA: Principles of Service Design

ISBN 9780132344821

Published with over 240 color illustrations, this hands-on guide contains practical, comprehensive, and in-depth coverage of service engineering techniques and the service-orientation design paradigm. Proven design principles are documented to help maximize the strategic benefit potential of SOA.



SOA: Design Patterns

ISBN 9780136135166

Software design patterns have emerged as a powerful means of avoiding and overcoming common design problems and challenges. This new book presents a formal catalog of design patterns specifically for SOA and service-orientation. All patterns are documented using full-color illustrations and further supplemented with case study examples.

Several additional series titles are currently in development and will be released soon. For more information about any of the books in this series, visit www.soabooks.com.

SOA & Cloud Computing Training & Certification



Content from this book and other series titles has been incorporated into the SOA Certified Professional (SOACP) program, an industry-recognized, vendor-neutral SOA certification curriculum developed by author Thomas Erl in cooperation with industry experts and academic communities and provided by SOASchool.com and licensed training partners.

The SOA Certified Professional curriculum is comprised of a collection of 23 courses and labs that can be taken with or without formal testing and certification. Training can be delivered anywhere in the world by certified instructors. A comprehensive self-study program is available for remote, self-paced study, and exams can be taken world-wide via Prometric testing centers.

Certifications include:

- Certified SOA Professional
- Certified SOA Architect
- Certified SOA Analyst
- Certified SOA Consultant
- Certified SOA Java Developer
- Certified SOA .NET Developer
- Certified SOA Governance Specialist
- Certified SOA Security Specialist
- Certified SOA Quality Assurance Specialist

All courses are reviewed and revised on a regular basis to stay in alignment with industry developments

For more information, visit: www.soaschool.com



PROMETRIC



SOASchool.com and CloudSchool.com exams offered world-wide through Prometric testing centers (www.prometric.com).



The Cloud Certified Professional (CCP) program, provided by CloudSchool.com, establishes a series of vendor-neutral industry certifications dedicated to areas of specialization in the field of cloud computing. Also founded by author Thomas Erl, this program exists independently from the SOASchool.com courses, while preserving consistency in terminology, conventions, and notation. This allows IT professionals to study cloud computing topics separately or in combination with SOA topics, as required.

The Cloud Certified Professional curriculum is comprised of 15 courses and labs, each of which has a corresponding Prometric exam. Private and public training workshops can be provided throughout the world by certified instructors. Self-study kits are further available for remote, self-paced study and in support of instructor-led workshops.

Certifications include:

- Certified Cloud Professional
- Certified Cloud Technology Professional
- Certified Cloud Architect
- Certified Cloud Governance Specialist
- Certified Cloud Security Specialist
- Certified Cloud Storage Specialist

All courses are reviewed and revised on a regular basis to stay in alignment with industry developments

For more information, visit: www.cloudschool.com



Self-Study Kits available for remote, self-paced study and exam preparation.



