Software Systems Architecture

Second Edition

Working With Stakeholders Using Viewpoints and Perspectives

NICK ROZANSKI · EOIN WOODS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Software Systems Architecture

Second Edition

This page intentionally left blank

Software Systems Architecture

Working with Stakeholders Using Viewpoints and Perspectives

Second Edition

Nick Rozanski
Eoin Woods

Addison-Wesley

Upper Saddle River, NJ ▪ Boston ▪ Indianapolis ▪ San Francisco
New York ▪ Toronto ▪ Montreal ▪ London ▪ Munich ▪ Paris ▪ Madrid
Capetown ▪ 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: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Rozanski, Nick.
Software systems architecture : working with stakeholders using viewpoints and perspectives / Nick Rozanski, Eoin Woods. 2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-321-71833-X (hard cover : alk. paper)
1. Computer software Development. 2. Computer architecture. I. Woods, Eoin. II. Title.

QA76.76.D47R696 2012
005.3 dc23 2011040253

Copyright © 2012 Pearson Education, 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. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-71833-4
ISBN-10: 0-321-71833-X
Text printed in the United States on recycled paper at Courier in Westford, Massachusetts.
Second printing, September 2012

To my family, Isabel, Sophie, Alex, and Luci
—NR

To my parents, Anne and Desmond,
and to my family, Lynda and Katherine
—EW

This page intentionally left blank

CONTENTS

PREFACE TO THE SECOND EDITION xv
Acknowledgments for the Second Edition xvi

PREFACE TO THE FIRST EDITION xvii
Acknowledgments xx

CHAPTER 1 INTRODUCTION 1
Stakeholders, Viewpoints, and Perspectives 1
The Structure of This Book 7
Who Should Read This Book 7
Conventions Used 8

PART I ARCHITECTURE FUNDAMENTALS 9

CHAPTER 2 SOFTWARE ARCHITECTURE CONCEPTS 11
Software Architecture 11
Architectural Elements 20
Stakeholders 21
Architectural Descriptions 24
Relationships between the Core Concepts 26
Summary 27
Further Reading 28

CHAPTER 3 VIEWPOINTS AND VIEWS 31
Architectural Views 34
Viewpoints 36
Relationships between the Core Concepts 37
The Benefits of Using Viewpoints and Views 38
Viewpoint Pitfalls 39
Our Viewpoint Catalog 39

Summary 43
Further Reading 43

CHAPTER 4 ARCHITECTURAL PERSPECTIVES 45
Quality Properties 45
Architectural Perspectives 47
Applying Perspectives to Views 51
Consequences of Applying a Perspective 54
Relationships between the Core Concepts 56
The Benefits of Using Perspectives 56
Perspective Pitfalls 58
Comparing Perspectives to Viewpoints 58
Our Perspective Catalog 60
Summary 61
Further Reading 62

CHAPTER 5 THE ROLE OF THE SOFTWARE ARCHITECT 63
The Architecture Definition Process 64
The Role of the Architect 68
Interrelationships between the Core Concepts 71
Architectural Specializations 72
The Organizational Context 73
The Architect's Skills 76
The Architect's Responsibilities 77
Summary 78
Further Reading 79

PART II THE PROCESS OF SOFTWARE ARCHITECTURE 81
CHAPTER 6 INTRODUCTION TO THE SOFTWARE ARCHITECTURE PROCESS 83

CHAPTER 7 THE ARCHITECTURE DEFINITION PROCESS 85
Guiding Principles 85
Process Outcomes 86
The Process Context 87
Supporting Activities 89
Architecture Definition Activities 92
Process Exit Criteria 97
Architecture Definition in the Software Development Lifecycle 98
Summary 102
Further Reading 103

CHAPTER 8 CONCERNS, PRINCIPLES, AND DECISIONS 105
Problem-Focused Concerns 108
Solution-Focused Concerns 111

Other Real-World Constraints 114
What Makes a Good Concern 116
Architectural Principles 117
Architectural Decisions 122
Using Principles to Link Concerns and Decisions 125
Checklist 128
Summary 128
Further Reading 129

CHAPTER 9 IDENTIFYING AND ENGAGING STAKEHOLDERS 131
Selection of Stakeholders 131
Classes of Stakeholders 133
Examples 138
Proxy Stakeholders 140
Stakeholder Groups 141
Stakeholders' Responsibilities 141
Checklist 142
Summary 142
Further Reading 143

CHAPTER 10 IDENTIFYING AND USING SCENARIOS 145
Types of Scenarios 146
Uses for Scenarios 147
Identifying and Prioritizing Scenarios 148
Capturing Scenarios 149
What Makes a Good Scenario? 153
Applying Scenarios 154
Effective Use of Scenarios 157
Checklist 159
Summary 159
Further Reading 160

CHAPTER 11 USING STYLES AND PATTERNS 161
Introducing Design Patterns 161
Styles, Patterns, and Idioms 164
Patterns and Architectural Tactics 166
An Example of an Architectural Style 167
The Benefits of Using Architectural Styles 170
Styles and the Architectural Description 172
Applying Design Patterns and Language Idioms 172
Checklist 174
Summary 174
Further Reading 175

Chapter 12 Producing Architectural Models 177
Why Models Are Important 178
Types of Models 181
Modeling Languages 184
Guidelines for Creating Effective Models 187
Modeling with Agile Teams 193
Checklist 194
Summary 195
Further Reading 196

Chapter 13 Creating the Architectural Description 197
Properties of an Effective Architectural Description 198
Glossaries 206
The ISO Standard 206
Contents of the Architectural Description 207
Presenting the Architectural Description 213
Checklist 215
Summary 216
Further Reading 216

Chapter 14 Evaluating the Architecture 217
Why Evaluate the Architecture? 218
Evaluation Techniques 219
Scenario-Based Evaluation Methods 226
Evaluation during the Software Lifecycle 230
Validating the Architecture of an Existing System 233
Recording the Results of Evaluation 236
Choosing an Evaluation Approach 237
Checklist 238
Summary 238
Further Reading 239

Part III A Viewpoint Catalog 241
Chapter 15 Introduction to the Viewpoint Catalog 243

Chapter 16 The Context Viewpoint 247
Concerns 248
Models 255
Problems and Pitfalls 261
Checklist 265
Further Reading 266

CHAPTER 17 THE FUNCTIONAL VIEWPOINT 267
Concerns 268
Models 271
Problems and Pitfalls 285
Checklist 291
Further Reading 292

CHAPTER 18 THE INFORMATION VIEWPOINT 293
Concerns 294
Models 311
Problems and Pitfalls 322
Checklist 330
Further Reading 330

CHAPTER 19 THE CONCURRENCY VIEWPOINT 333
Concerns 335
Models 340
Problems and Pitfalls 351
Checklist 355
Further Reading 355

CHAPTER 20 THE DEVELOPMENT VIEWPOINT 357
Concerns 358
Models 360
Problems and Pitfalls 367
Checklist 370
Further Reading 371

CHAPTER 21 THE DEPLOYMENT VIEWPOINT 373
Concerns 374
Models 378
Problems and Pitfalls 387
Checklist 391
Further Reading 392

CHAPTER 22 THE OPERATIONAL VIEWPOINT 393
Concerns 394
Models 402
Problems and Pitfalls 419
Checklist 423
Further Reading 424

CHAPTER 23 ACHIEVING CONSISTENCY ACROSS VIEWS 425
Relationships between Views 426
Context and Functional View Consistency 427
Context and Information View Consistency 427
Context and Deployment View Consistency 428
Functional and Information View Consistency 428
Functional and Concurrency View Consistency 429
Functional and Development View Consistency 430
Functional and Deployment View Consistency 430
Functional and Operational View Consistency 431
Information and Concurrency View Consistency 431
Information and Development View Consistency 432
Information and Deployment View Consistency 432
Information and Operational View Consistency 432
Concurrency and Development View Consistency 433
Concurrency and Deployment View Consistency 433
Deployment and Operational View Consistency 434

PART IV THE PERSPECTIVE CATALOG 435

CHAPTER 24 INTRODUCTION TO THE PERSPECTIVE CATALOG 437

CHAPTER 25 THE SECURITY PERSPECTIVE 439
Applicability to Views 441
Concerns 442
Activities: Applying the Security Perspective 446
Architectural Tactics 456
Problems and Pitfalls 465
Checklists 473
Further Reading 474

CHAPTER 26 THE PERFORMANCE AND SCALABILITY PERSPECTIVE 475
Applicability to Views 476
Concerns 476
Activities: Applying the Performance and Scalability Perspective 482
Architectural Tactics 491
Problems and Pitfalls 502
Checklists 509
Further Reading 510

CHAPTER 27 THE AVAILABILITY AND RESILIENCE PERSPECTIVE 511
Applicability to Views 512
Concerns 512
Activities: Applying the Availability and Resilience Perspective 516

Architectural Tactics 526
Problems and Pitfalls 533
Checklists 539
Further Reading 541

CHAPTER 28 THE EVOLUTION PERSPECTIVE 543
Applicability to Views 544
Concerns 545
Activities: Applying the Evolution Perspective 549
Architectural Tactics 552
Problems and Pitfalls 560
Checklists 564
Further Reading 565

CHAPTER 29 OTHER PERSPECTIVES 567
The Accessibility Perspective 568
The Development Resource Perspective 573
The Internationalization Perspective 579
The Location Perspective 585
The Regulation Perspective 591
The Usability Perspective 595

PART V PUTTING IT ALL TOGETHER 603

CHAPTER 30 WORKING AS A SOFTWARE ARCHITECT 605
Architecture in the Project Lifecycle 605
Supporting Different Types of Projects 615

APPENDIX OTHER VIEWPOINT SETS 621
Kruchten “4+1” 621
RM-ODP 623
Siemens (Hofmeister, Nord, and Soni) 623
SEI “Views and Beyond” Views 624
Garland and Anthony 626
IAF 627
Enterprise Architecture Frameworks 627
Other Enterprise Architecture Frameworks 629

BIBLIOGRAPHY 631
ABOUT THE AUTHORS 643
INDEX 645

This page intentionally left blank

Preface to the Second Edition

The IT landscape looks significantly different today from when we first started work on our book ten years ago. The world is a much more connected place, with computers and the Internet being a big part of many people's daily lives both at home and at work. This has led to an even greater expectation among users and other stakeholders that systems should be functionally rich and complete, easy to use, robust, scalable, and secure. We feel that the architect has an important role in achieving these goals and are heartened by the fact that this view seems to have gained fairly widespread acceptance among software development professionals and senior business and technology management.

We were delighted by the positive reception to the first edition of our book from practitioners, aspiring software architects, and academia. Our readers seemed to find it useful, comprehensive, and informative. However, architecture is a constantly changing discipline, and the second edition reflects what we have learned and improved upon in our own practice since the publication of the first edition. It also incorporates a number of very useful comments and suggestions for improvement from readers, for which we are extremely grateful.

However, our fundamental messages remain the same. Our primary focus is on architecture as a service to stakeholders and a way to ensure that an information system meets their needs. We continue to emphasize the vital importance of views as a way of representing an architecture's complexity in a way its stakeholders can understand. We are also unswerving in our belief that architecture must define how a system will provide the required quality properties—such as scalability, resilience, and security—as well as defining its static and dynamic structure, and that perspectives provide an effective way to do this.

Our main audience is practicing or aspiring architects, but we hope that other IT professionals, who may be working alongside an architect, and students, who will one day find themselves in this position, will also find it a useful read.

The most important changes in this edition are as follows.

- We have introduced a new viewpoint, which we call the Context viewpoint. This describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts). It extends, formalizes, and standardizes the relatively brief discussion of scope and context that used to be in Chapter 8.
- We have expanded the discussion of different aspects of the role of architecture in Part II.
- We have revised most of the viewpoint and perspective definitions, particularly the Functional and Concurrency views and the Performance and Scalability perspective.
- We have revised and extended the Bibliography and the Further Reading sections in most chapters.
- We have updated the book to align with the concepts and terminology in the new international architecture standard ISO 42010 (which derives from IEEE Standard 1471).
- We have updated our UML modeling advice and examples to reflect the changes introduced in version 2 of UML.

We hope that you find the second edition of the book a useful improvement and extension of the first edition, and we invite you to visit to our Web site at www.viewpoints-and-perspectives.info for further software architecture resources or to contact us to provide feedback on the book.

ACKNOWLEDGMENTS FOR THE SECOND EDITION

In addition to the people we thanked for the first edition, we would also like to thank our second-edition reviewers—Paul Clements, Tim Cull, Rich Hilliard, Philippe Kruchten, and Tommi Mikkonen—and our diligent and thorough copy editor, Barbara Wood. In particular, we would like to thank Paul for his thorough, insightful, and challenging comments and suggestions for improvement, which we found extremely useful.

Preface to the First Edition

The authors of this book are both practicing software architects who have worked in this role, together and separately, on information system development projects for quite a few years. During that time, we have seen a significant increase in the visibility of software architects and in the importance with which our role has been viewed by colleagues, management, and customers. No large software development project nowadays would expect to go ahead without an architect—or a small architectural group—in the vanguard of the development team.

While there may be an emerging consensus that the software architect's role is an important one, there seems to be little agreement on what the job actually involves. Who are our clients? To whom are we accountable? What are we expected to deliver? What is our involvement once the architectural design has been completed? And, perhaps most fundamentally, where are the boundaries between requirements, architecture, and design?

The absence of a clear definition of the role is all the more problematic because of the seriousness of the problems that today's software projects (and specifically, their architects) have to resolve.

- The expectations of users and other stakeholders in terms of functionality, capability, time to market, and flexibility have become much more demanding.
- Long system development times result in continual scope changes and consequent changes to the system's architecture and design.
- Today's systems are more functionally and structurally complex than ever and are usually constructed from a mix of off-the-shelf and custom-built components.

- Few systems exist in isolation; most are expected to interoperate and exchange information with many other systems.
- Getting the functional structure—the design—of the system right is only part of the problem. How the system behaves (i.e., its quality properties) is just as critical to its effectiveness as what it does.
- Technology continues to change at a pace that makes it very hard for architects to keep their technical expertise up-to-date.

When we first started to take on the role of software architects, we looked for some sort of software architecture handbook that would walk us through the process of developing an architectural design. After all, other architectural disciplines have behind them centuries of theory and established best practice.

For example, in the first century A.D., the Roman Marcus Vitruvius Pollio wrote the first ever architectural handbook, *De architectura libri decem* ("Ten Books on Architecture"), describing the building architect's role and required skills and providing a wealth of material on standard architectural structures. In 1670, Anthony Deane, a friend of diarist Samuel Pepys, a former mayor of the English town of Harwich, and later a member of Parliament, published a groundbreaking textbook, *A Doctrine of Naval Architecture*, which described in detail some of the leading methods of the time for large ship design. Deane's ideas and principles helped systematize the practice of naval architecture for many years. And in 1901, George E. Davis, a consulting engineer in the British chemical industry, created a new field of engineering when he published his text *A Handbook of Chemical Engineering*. This text was the first book to define the practical principles underpinning industrial chemical processes and guided the field for many years afterward.

The existence of such best practices has a very important consequence in terms of uniformity of approach. If you were to give several architects and engineers a commission to design a building, a cruise liner, or a chemical plant, the designs they produced would probably differ. However, the processes they used, the ways they represented their designs on paper (or a computer screen), and the techniques they used to ensure the soundness of their designs would be very similar.

Sadly, our profession has yet to build any significant legacy of mainstream industrial best practices. When we looked, we found a dearth of introductory books to guide practicing information systems architects in the details of doing their jobs.

Admittedly, we have an abundance of books on specific technologies, whether it's J2EE, CORBA, or .NET, and some on broader topics such as Web services or object orientation (although, because of the speed at which software technology changes, many of these become out-of-date within a few years). There are also a number of good general software architecture books, several of which we refer to in later chapters. But many of these books aim to lay down principles that apply across all sorts of systems and so are written in quite general terms, while most of the more specific texts are aimed at our colleagues in the real-time and embedded-systems communities.

We feel that if you are a new software architect for an information system, the books that actually tell you how to do your job, learn the important things you need to know, and make your architectural designs successful are few and far between. While we don't presume to replace the existing texts on software architecture or place ourselves alongside the likes of Vitruvius, Deane, and Davis, addressing these needs was the driving force behind our decision to write this book.

Specifically, the book shows you:

- What software architecture is about and why your role is vitally important to successful project delivery
- How to determine who is interested in your architecture (your *stakeholders*), understand what is important to them (their *concerns*), and design an *architecture* that reflects and balances their different needs
- How to communicate your architecture to your stakeholders in an understandable way that demonstrates that you have met their concerns (the *architectural description*)
- How to focus on what is *architecturally significant*, safely leaving other aspects of the design to your designers, without neglecting issues like performance, resilience, and location
- What important activities you most need to undertake as an architect, such as identifying and engaging stakeholders, using scenarios, creating models, and documenting and validating your architecture

Throughout the book we primarily focus on the development of large-scale information systems (by which we mean the computer systems used to automate the business operations of large organizations). However, we have tried to present our material in a way that is independent of the type of information system you are designing, the technologies the developers will be using, and the software development lifecycle your project is following. We have standardized on a few things, such as the use of Unified Modeling Language (UML) in most of our diagrams, but we've done that only because UML is the most widely understood modeling language around. You don't have to be a UML expert to understand this book.

We didn't set out to be the definitive guide to developing the architecture of your information system—such a book would probably never be finished and would require the collaboration of a huge number of experts across a wide range of technical specializations. Also, we did not write a book of prescriptive methods. Although we present some activity diagrams that explain how to produce your deliverables, these are designed to be compatible with the wide range of software development approaches in use today.

What we hope we have achieved is the creation of a practical, practitioner-oriented guide that explains how to design successful architectures for information systems and how to see these through to their successful implementation. This is the sort of book that we wish had been available when we started out as software architects, and one that we expect to refer to even now.

You can find further useful software architecture resources, and contact us to provide feedback on the book's content, via our Web page: www.viewpoints-and-perspectives.info. We look forward to hearing from you.

ACKNOWLEDGMENTS

This book would never have appeared without the advice, assistance, and support of a lot of people.

We are very grateful to the many reviewers who commented on the text at various stages of its creation, including Gary Birch, Chris Britton, Kelley Butler, Sholom Cohen, Dan Haywood, Sallie Henry, Andy Longshaw, Robert Nord, Dan Paulish, Martyn Thomas, and Hans van Vliet.

We'd also like to thank the team members at Addison-Wesley for all of their work to make the book a reality, including Kim Boedigheimer, John Fuller, Peter Gordon, Chrysta Meadowbrooke, Simon Plumtree, and Elizabeth Ryan.

Other people who provided us with advice, encouragement, and inspiration at various times include Felix Bachmann, Dave Borthwick, David Emery, Wolfgang Emmerich, Rich Hilliard, Philippe Kruchten, Roland Leibundgut, Mike Mackay, Dave Maher, Mark Maier, Lucia Rapanotti, and Gaynor Redvers-Mutton.

We would also like to thank our families for their constant love, encouragement, and support throughout the project.

3

Viewpoints and Views

When you start the daunting task of designing the architecture of your system, you will find that you have some difficult architectural questions to answer.

- What are the main functional elements of your architecture?
- How will these elements interact with one another and with the outside world?
- What information will be managed, stored, and presented?
- What physical hardware and software elements will be required to support these functional and information elements?
- What operational features and capabilities will be provided?
- What development, test, support, and training environments will be provided?

A common temptation—one you should strongly avoid—is to try to answer all of these questions by means of a single, heavily overloaded, all-encompassing model. This sort of model (and we've all seen them) will probably use a mixture of formal and informal notations to describe a number of aspects of the system on one huge sheet of paper: the functional structure, software layering, concurrency, intercomponent communication, physical deployment environment, and so on. Let's see what happens when we try to use an all-encompassing model in our AD, by means of an example.

As the example shows, this sort of AD is really the worst of all worlds. Many writers on software architecture have pointed out that it simply isn't possible to describe a software architecture by using a single model. Such a model is hard to understand and is unlikely to clearly identify the architecture's most

EXAMPLE Although the airline reservation system we introduced in Chapter 2 is conceptually fairly simple, in practice some aspects of this system make it very complicated indeed.

- The system's data is distributed across a number of systems in different physical locations.
- A number of different types of data entry devices must be supported.
- The system must be able to present some information in different languages.
- The system must be able to print tickets and other documents on a wide range of printers.
- The plethora of international regulations complicates the picture even further.

After some discussion, the architect draws up a first-cut architecture for the system, which attempts to represent all of its important aspects in a single diagram. This model includes the full range of data entry devices (including various dumb terminals, desktop PCs, and wireless devices), the multiple physical systems on which data is stored or replicated data is maintained, and some of the printing devices that must be supported (the model does not cover remote printing because it is done at a separate facility). The model is heavily annotated with text to indicate, for example, where multilanguage support is required and where data must be audited, archived, or analyzed to support regulatory requirements.

However, no details of the network interfaces between the different components are included—these are abstracted out into a network icon because they are so complex. (In fact, the network design is probably the most complicated aspect of the architecture, requiring support for a number of different and largely incompatible network protocols, routing over public and private networks, synchronous and asynchronous interactions, and varying levels of service reliability and availability.) Furthermore, the model does not address any of the implications of having the same data distributed around multiple systems.

Because it is so complex and tries to address a wide mix of concerns in the same diagram, the model fails to engage any of the stakeholders. The users find it too complex and difficult to understand (particularly because of the large number of physical hardware components represented). The technology stakeholders, on the other hand, tend to disregard it because of the detail that is left out, such as the network topology. The legal team members can't use it to satisfy themselves that the regulatory aspects will be adequately handled, and the sponsor finds it completely incomprehensible.

Furthermore, the architect spends an inordinate amount of time keeping it up-to-date—every time a new type of data entry device or printer is discussed, for example, the diagram needs to be updated and reprinted on a very large sheet of paper.

Because of these problems, the diagram soon becomes obsolete and is eventually forgotten. Unfortunately, the issues that the model fails to address do not disappear and thus cause many problems and delays during the implementation and the early stages of live operation.

important features. It tends to poorly serve individual stakeholders because they struggle to understand the aspects that interest them. Worst of all, because of its complexity, a monolithic AD is often incomplete, incorrect, or out-of-date.

Principle It is not possible to capture the functional features and quality properties of a complex system in a single comprehensible model that is understandable by, and of value to, its stakeholders.

We need to represent complex systems in a way that is manageable and comprehensible by a range of business and technical stakeholders. A widely used approach—the only successful one we have found—is to attack the problem from different directions simultaneously. In this approach, the AD is partitioned into a number of separate but interrelated *views*, each of which describes a separate aspect of the architecture. Collectively, the views describe the whole system.

To help you understand what we mean by a view, let's consider the example of an architectural drawing for one of the elevations of an office block. This portrays the building from a particular aspect, typically a compass bearing such as northeast. The drawing shows features of the building that are visible from that vantage point but not from other directions. It doesn't show any details of the interior of the building (as seen by its occupants) or of its internal systems (such as plumbing or air conditioning) that influence the environment its occupants will inhabit. Thus the blueprint is only a partial representation of the building; you have to look at—and understand—the whole set of blueprints to grasp the facilities and experience that the whole building will provide.

Another way that a building architect might represent a new building is to construct a scale model of it and its environs. This shows how the building will look from all sides but again reveals nothing about the mechanisms to be used in its construction, its interior form, or its likely internal environment.

Strategy A complex system is much more effectively described by a set of interrelated views, which collectively illustrate its functional features and quality properties and demonstrate that it meets its goals, than by a single overloaded model.

Let's take a look at what this approach means for software architecture.

Architectural Views

An architectural view is a way to portray those aspects or elements of the architecture that are relevant to the concerns the view intends to address—and, by implication, the stakeholders to whom those concerns are important.

This idea is not new, going back at least as far as the work of David Parnas in the 1970s and more recently Dewayne Perry and Alexander Wolf in the early 1990s. However, it wasn't until 1995 that Philippe Kruchten of the Rational Corporation published his widely accepted written description of views, *Architectural Blueprints—The "4 + 1" View Model of Software Architecture*. This suggested four different views of a system and the use of a set of scenarios (use cases) to elucidate its behavior. Kruchten's approach has since evolved to form an important part of the Rational Unified Process (RUP).

IEEE Standard 1471 (the predecessor of ISO Standard 42010) formalized these concepts in 2000 and brought some welcome standardization of terminology. In fact, our definition of a view is based on and extends the one from the original IEEE standard.

Definition A **view** is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders.

When deciding what to include in a view, ask yourself the following questions.

- *View scope*: What structural aspects of the architecture are you trying to represent? For example, are you trying to define the runtime functional elements and their intercommunication, or the runtime environment and how the system is deployed into it? Do you need to represent the dynamic or static elements of these structures? (For example, in the case of the functional element structure, do you wish to show the elements and the connectors between them, or the sequence of interactions they perform in order to process an incoming request, or both?)

- *Element types*: What type(s) of architectural element are you trying to categorize? For example, when considering how the system is deployed, do you need to represent individual server machines, or do you just need to represent a service environment (like Force.com SiteForce or Google AppEngine) that your system elements are deployed into?
- *Audience*: What class(es) of stakeholder is the view aimed at? A view may be narrowly focused on one class of stakeholder or even a specific individual, or it may be aimed at a larger group whose members have varying interests and levels of expertise.
- *Audience expertise*: How much technical understanding do these stakeholders have? Acquirers and users, for example, will be experts in their subject areas but are unlikely to know much about hardware or software, while the converse may apply to developers or support staff.
- *Scope of concerns*: What stakeholder concerns is the view intended to address? How much do the stakeholders know about the architectural context and background to these concerns?
- *Level of detail*: How much do these stakeholders need to know about this aspect of the architecture? For nontechnical stakeholders such as users, how competent are they in understanding its technical details?

As with the AD itself, one of your main challenges is to get the right content into your views. Provide too much irrelevant detail, for example, and your audience will be overwhelmed; too little information, and you risk your audience being confused or making assumptions that may not be valid. There are two key questions you should ask yourself when deciding what to include in a view. First of all, can the stakeholders that it targets use it to determine whether their concerns have been met? And second, can those stakeholders use it to successfully undertake their role in building the system?

We will explore the second question in more detail in Chapter 9, but for now we will summarize these questions as follows.

STRATEGY Only include in a view information that furthers the objectives of your AD—that is, information that helps explain the architecture to stakeholders or demonstrates that the goals of the system (i.e., the concerns of its stakeholders) are being met.

Viewpoints

It would be hard work if every time you were creating a view of your architecture you had to go back to first principles to define what should go into it. Fortunately, you don't quite have to do that.

In his introductory paper, Philippe Kruchten defined four standard views, namely, Logical, Process, Physical, and Development. The IEEE standard made this idea generic (and did not specify one set of views or another) by proposing the concept of a *viewpoint*.

The objective of the viewpoint concept is an ambitious one—no less than making available a library of templates and patterns that can be used off the shelf to guide the creation of an architectural view that can be inserted into an AD. We define a viewpoint (again after IEEE Standard 1471) as follows.

Definition A **viewpoint** is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views.

Architectural viewpoints provide a framework for capturing reusable architectural knowledge that can be used to guide the creation of a particular type of (partial) AD. You may find it helpful to compare the relationship between viewpoints and views to the relationship between classes and objects in object-oriented development.

- A class definition provides a template for the construction of an object. An object-oriented system will include at runtime a number of *objects*, each of a specified *class*.
- A viewpoint provides a template for the construction of a view. A viewpoints-and-views-based architecture definition will include a number of *views*, each conforming to a specific *viewpoint*.

Viewpoints are an important way of bringing much-needed structure and consistency to what was in the past a fairly unstructured activity. By defining a standard approach, a standard language, and even a standard metamodel for describing different aspects of a system, stakeholders can understand any AD that conforms to these standards once familiar with them.

In practice, of course, we haven't fully achieved this goal yet. There are no universally accepted ways to model software architectures, and many ADs use their own homegrown conventions (or even worse, no particular conventions at all). However, the widespread acceptance of techniques such as entity-relationship models and of modeling languages such as UML takes us some way toward this goal.

In any case, it is extremely useful to be able to categorize views according to the types of concerns and architectural elements they present.

STRATEGY When developing a view, whether or not you use a formally defined viewpoint, be clear in your own mind what sorts of concerns the view is addressing, what types of architectural elements it presents, and who the viewpoint is aimed at. Make sure that your stakeholders understand these as well.

RELATIONSHIPS BETWEEN THE CORE CONCEPTS

To put views and viewpoints in context, we can now extend the conceptual model we introduced in Chapter 2 to illustrate how views and viewpoints contribute to the overall picture (see Figure 3–1).

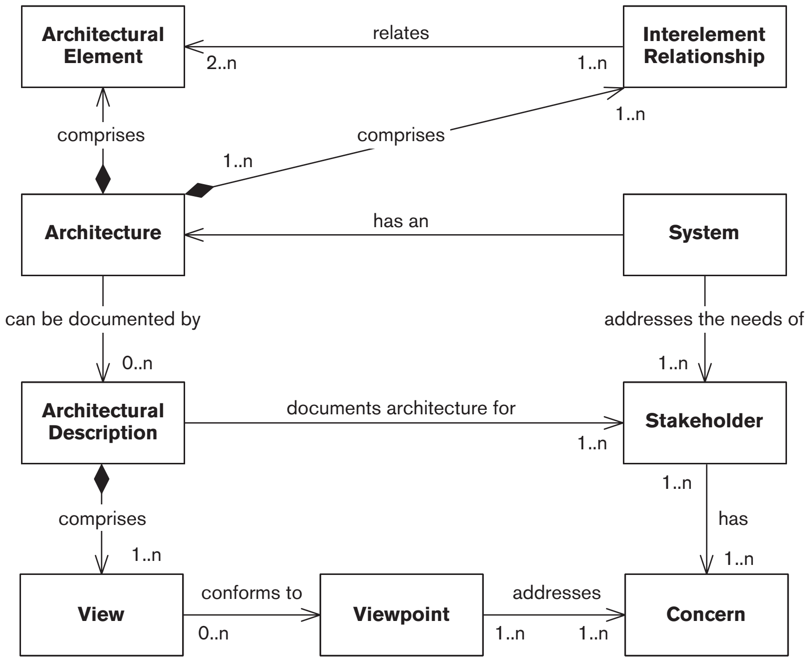

FIGURE 3–1 VIEWS AND VIEWPOINTS IN CONTEXT

We have added the following relationships to the diagram we originally presented as Figure 2–5.

- A viewpoint defines the aims, intended audience, and content of a class of views and defines the concerns that views of this class will address.
- A view conforms to a viewpoint and so communicates the resolution of a number of concerns (and a resolution of a concern may be communicated in a number of views).
- An AD comprises a number of views.

THE BENEFITS OF USING VIEWPOINTS AND VIEWS

Using views and viewpoints to describe the architecture of a system benefits the architecture definition process in a number of ways.

- *Separation of concerns:* Describing many aspects of the system via a single representation can cloud communication and, more seriously, can result in independent aspects of the system becoming intertwined in the model. Separating different models of a system into distinct (but related) descriptions helps the design, analysis, and communication processes by allowing you to focus on each aspect separately.
- *Communication with stakeholder groups*: The concerns of each stakeholder group are typically quite different (e.g., contrast the primary concerns of end users, security auditors, and help-desk staff), and communicating effectively with the various stakeholder groups is quite a challenge. The viewpoint-oriented approach can help considerably with this problem. Different stakeholder groups can be guided quickly to different parts of the AD based on their particular concerns, and each view can be presented using language and notation appropriate to the knowledge, expertise, and concerns of the intended readership.
- *Management of complexity*: Dealing simultaneously with all of the aspects of a large system can result in overwhelming complexity that no one person can possibly handle. By treating each significant aspect of a system separately, the architect can focus on each in turn and so help conquer the complexity resulting from their combination.
- *Improved developer focus*: The AD is of course particularly important for the developers because they use it as the foundation of the system design. By separating out into different views those aspects of the system that are particularly important to the development team, you help ensure that the right system gets built.

VIEWPOINT PITFALLS

Of course, the use of views and viewpoints won't solve all of your software architecture problems automatically. Although we have found that using views is really the only way to make the problem manageable, you need to be aware of some possible pitfalls when using the view-and-viewpoint-based approach.

- *Inconsistency*: Using a number of views to describe a system inevitably brings consistency problems. It is theoretically possible to use architecture description languages to create the models in your views and then cross-check these automatically (much as graphical modeling tools attempt to check structured or object-oriented methods models), but there are no such machine-checkable architecture description languages in widespread use today. This means that achieving cross-view consistency within an AD is an inherently manual process. To assist with this, Chapter 23 includes a checklist to help you ensure consistency between the standard viewpoints presented in our catalog in Part III.
- *Selection of the wrong set of views*: It is not always obvious which set of views is suitable for describing a particular system. This is influenced by a number of factors, such as the nature and complexity of the architecture, the skills and experience of the stakeholders (and of the architect), and the time available to produce the AD. There really isn't an easy answer to this problem, other than your own experience and skill and an analysis of the most important concerns that affect your architecture.
- *Fragmentation*: Having several views of your architecture can make the AD difficult to understand. Each separate view also involves a significant amount of effort to create and maintain. To avoid fragmentation and minimize the overhead of maintaining unnecessary descriptions, you should eliminate views that do not address significant concerns for the system you are building. In some cases, you may also consider creating hybrid views that combine models from a number of views in the viewpoint set (e.g., creating a combined deployment and concurrency view). Beware, however, of the combined views becoming difficult to understand and maintain because they address a combination of concerns.

OUR VIEWPOINT CATALOG

Part III of this book presents our catalog of seven core viewpoints for information systems architecture: the Context, Functional, Information, Concurrency, Development, Deployment, and Operational viewpoints. Although the viewpoints are (largely) disjoint, we find it convenient to group them as shown in Figure 3–2.

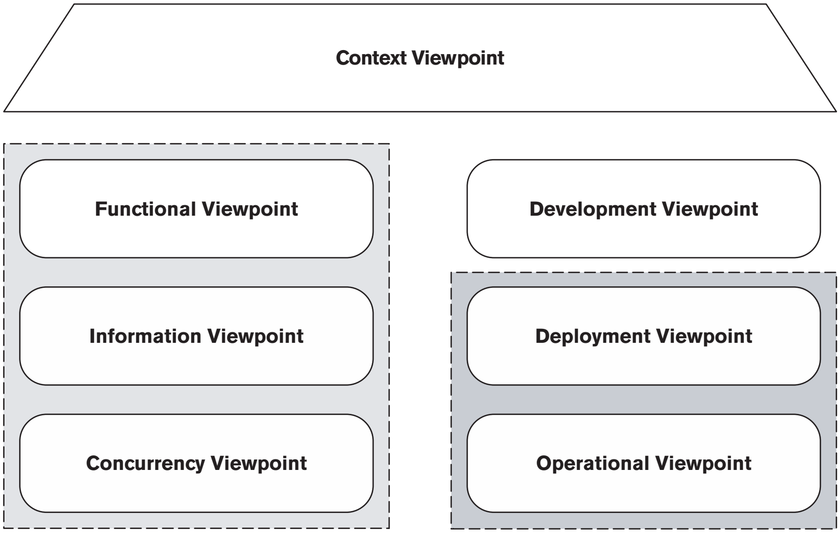

FIGURE 3–2 VIEWPOINT GROUPINGS

- The Context viewpoint describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts).
- The Functional, Information, and Concurrency viewpoints characterize the fundamental organization of the system.
- The Development viewpoint exists to support the system's construction.
- The Deployment and Operational viewpoints characterize the system once in its live environment.

You can use the shape and position of the icons in Figure 3–2 to help understand how our viewpoints are related to one another. We have put the Context viewpoint at the top of the diagram to indicate its role as the "overarching" viewpoint that informs the scope and content of all the others. We group the Functional, Information, and Concurrency viewpoints together at the left, to highlight that between them they define how the system provides its functionality.

The viewpoints on the right-hand side are to some extent driven by those on the left; for example, the Development viewpoint defines standards and models for the construction of the architecture's functional, information, and concurrency elements. We have further grouped the Deployment and Operational viewpoints, since between them, these views define the system's production environment.

Viewpoint Overview

Table 3–1 briefly describes our viewpoints.

Of course, not all of these viewpoints may apply to your architecture, and some will be more important than others. You may not need views of all of these types in your AD, and in some cases there may be other viewpoints that you need to identify and add yourself. This means that your first job is to understand the nature of your architecture, the skills and experience of the stakeholders, and the time available and other constraints, and then to come up with an appropriate selection of views.

FIGURE 3–2 VIEWPOINT CATALOG

Viewpoint	**Definition**
Context	Describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts). The Context view will be of interest to many of the system's stakeholders and plays an important role in helping them to understand its responsibilities and how it relates to their organization.
Functional	Describes the system's runtime functional elements, their responsibilities, interfaces, and primary interactions. A Functional view is the cornerstone of most ADs and is often the first part of the description that stakeholders try to read. It drives the shape of other system structures such as the information structure, concurrency structure, deployment structure, and so on. It also has a significant impact on the system's quality properties such as its ability to change, its ability to be secured, and its runtime performance.
Information	Describes the way that the system stores, manipulates, manages, and distributes information. The ultimate purpose of virtually any computer system is to manipulate information in some form, and this viewpoint develops a complete but high-level view of static data structure and information flow. The objective of this analysis is to answer the big questions around content, structure, ownership, latency, references, and data migration.
Concurrency	Describes the concurrency structure of the system and maps functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently and how this is coordinated and controlled. This entails the creation of models that show the process and thread structures that the system will use and the interprocess communication mechanisms used to coordinate their operation.
Development	Describes the architecture that supports the software development process. Development views communicate the aspects of the architecture of interest to those stakeholders involved in building, testing, maintaining, and enhancing the system.

Continued on next page

FIGURE 3–2 VIEWPOINT CATALOG *(CONTINUED)*

Viewpoint	Definition
Deployment	Describes the environment into which the system will be deployed and the dependencies that the system has on elements of it. This view captures the hardware environment that your system needs (primarily the processing nodes, network interconnections, and disk storage facilities required), the technical environment requirements for each element, and the mapping of the software elements to the runtime environment that will execute them.
Operational	Describes how the system will be operated, administered, and supported when it is running in its production environment. For all but the simplest systems, installing, managing, and operating the system is a significant task that must be considered and planned at design time. The aim of the Operational viewpoint is to identify system-wide strategies for addressing the operational concerns of the system's stakeholders and to identify solutions that address these.

While it can be hard to generalize, and it is important to choose your set of views for the specific context in which you find yourself, Table 3–2 lists the relative importance that we have often found each view to have for some typical types of information systems. We suggest you use this table as a starting point when choosing the views to include in your AD.

TABLE 3–2 MOST IMPORTANT VIEWS FOR TYPICAL SYSTEM TYPES

	OLTP Information System	Calculation Service/ Middleware	DSS/MIS System	High-Volume Web Site	Enterprise Package
Context	High	Low	High	Medium	Medium
Functional	High	High	Low	High	High
Information	Medium	Low	High	Medium	Medium
Concurrency	Low	High	Low	Medium	Varies
Development	High	High	Low	High	High
Deployment	High	High	High	High	High
Operational	Varies	Low	Medium	Medium	High

Summary

Capturing the essence and the detail of the whole architecture in a single model is just not possible for anything other than simple systems. If you try to do this, you will end up with a Frankenstein monster of a model that is unmanageable and does not adequately represent the system to you or any of the stakeholders.

By far the best way of managing this complexity is to produce a number of different representations of all or part of the architecture, each of which focuses on certain aspects of the system, showing how it addresses some of the stakeholder concerns. We call these *views*.

To help you decide what views to produce and what should go into any particular view, you use *viewpoints*, which are standardized definitions of view concepts, content, and activities.

The use of views and viewpoints brings many benefits, such as separation of concerns, improved communication with stakeholders, and management of complexity. However, it is not without its pitfalls, such as inconsistency and fragmentation, and you must be careful to manage these.

In this chapter, we introduced our viewpoint catalog, comprising the Context, Functional, Information, Concurrency, Development, Deployment, and Operational viewpoints, which we describe in detail in Part III.

Further Reading

A lot of useful guidance on creating ADs using views (including a discussion of when and how to combine views) and thorough guidance for creating the documentation for a wide variety of types of views can be found in Clements et al. [CLEM10]. Other references that help to make sense of viewpoints and views are IEEE Standard 1471 [IEEE00], ISO Standard 42010 [ISO11], and Kruchten's “4 + 1” approach [KRUC95]. One of the earliest explicit references to the need for architectural views appears in Perry and Wolf [PERR92].

Some of the other viewpoint taxonomies that have been developed over the last decade or so—including Kruchten's “4 + 1,” RM-ODP, the viewpoint set by Hofmeister et al. [HOFM00], and the set by Garland and Anthony [GARL03]—are described in the Appendix, together with recommendations for further reading in this area.

Part III, where we describe our viewpoint catalog in detail, contains references for specific view-related reading.

This page intentionally left blank

INDEX

Numbers
"4+1" viewpoint set (Krutchen), 621–622

A
Abstraction
  care and precision in the use of, 189
  facilitating change, 555
  in IAF (Integrated Architecture Framework), 627
  as modeling skill, 179
  in SEI viewpoint catalog, 625
  using for precision, 205
  validation of, 218
Acceptance criteria, in process outcomes, 87
Access control
  authentication. *See* Authentication
  authorization. *See* Authorization
  ensuring information secrecy, 460
  insider threats and, 469–470
  principles, 456–458
  resources and, 440
  security policies and, 449
Accessibility perspective
  activities, 570–571
  applicability to views, 569–570
  architectural tactics, 571–572
  concerns, 570
  defined, 568
  desired quality, 438
  further reading, 572–573
  overview of, 568–569
  problems and pitfalls, 572
  review checklists, 572–573
Accountability
  concerns of Security perspective, 444
  ensuring information secrecy, 462
  security policy for, 448
ACID (Atomic, Consistent, Isolated, and Durable) transaction properties, 302–303
Acquirers, classes of stakeholders, 133, 135
Action entities, in state model, 348
Actors (participants), representing in UML use cases, 198
AD (architectural description)
  architectural styles and, 172
  breaking complex system into interrelated views, 33–34
  checklist for, 215
  creating, 197–198
  defined, 24, 92, 197
  documenting, 177
  ISO standard and, 206–207
  limitations of monolithic models, 32–33
  overburdening, 368
  overview of, 24–26
  presenting, 213–215
  relationships with core concepts, 26–27
  sharing models and, 59
  summary and further reading, 216
  views in, 177–178
AD (architectural description) contents
  appendices, 212–213
  design decisions, 209–210
  Document Control section, 208
  Glossary, 206
  Introduction and Management Summary section, 209
  issues to be addressed, 212
  overview of, 207–208
  principles, 209
  quality properties, 211
  scenarios, 211
  Stakeholders section, 209
  Table of Contents, 208
  views and viewpoints, 210–211

AD (architectural description) properties
clarity, 203–204
conciseness, 201–203
correctness, 198–199
currency, 204
overview of, 198
precision, 205
sufficiency, 199–200
timeliness, 200–201
Ad hoc diagrams, 485
Ad hoc release management, 563–564
Adaptation, benefits of architectural styles, 170
Adapter design pattern, 162
ADLs (architecture description languages), 184–185, 276
ADM (Architecture Development Method), of TOGAF, 628
Administration
ensuring adequate facilities for, 466–467
provide security, 464
Administration models, 409–414
Agile Manifesto, 101, 607
Agile methods
deferring decision to “last responsible moment,” 201
overview of, 193–194
plan-driven methods compared with, 610
in Software development lifecycle, 100–102
team approach in, 193–194
Agile projects, 607–609
Aging, of information, 308–309
Agreements/contracts, evaluation as tool for creating, 219
Alert notifications
alert starvation or alert flooding, 423
integrating with third-party hosting environments and, 401
overview of, 397–398
Allocation styles, in SEI viewpoint catalog, 625
Americans with Disabilities Act, United States, 570
Appendices, 205, 212–213
Application code, avoiding embedding security in, 471–472
Architect. *See* Software architects
Architectural description. *See* AD (architectural description)
Architectural elements. *See* Elements
Architectural models. *See* Models
Architectural perspective. *See* Perspectives
Architectural styles or pattern. *See* Styles
Architectural tactics. *See* Tactics
Architectural views. *See* Views
Architecturally significance, 67–68, 124–125
Architecture definition
activities, 92–96
Agile methods in software development, 100–102
aspects of, 64
boundary between design and, 67–68
boundary between requirements analysis and, 66
evaluation techniques for system construction phase of lifecycle, 231–232
guiding principles, 85–86
interrelationship with core concepts, 71–72
ISO standard 42010 for, 58
iterative approaches to software development, 100
models in, 178
overview of, 85
process context, 87–89
process exit criteria, 97–98
process outcomes, 86–87
scenarios providing input to, 147
security administration provided as part of, 464
separating design from requirements analysis, 65–66
in software development lifecycle, 98
styles in, 170
summary and further reading, 102–103
supporting activities in, 89–92
timeliness of, 200
waterfall approaches to software development, 99–100
Architecture description languages (ADLs), 184–185, 276
Architecture Development Method (ADM), of TOGAF, 628

Architecture Tradeoff Analysis Method. *See* ATAM (Architecture Tradeoff Analysis Method)
Archiving/retaining information, 309–310
Artifacts, 56, 379–380
ARTS Standard Relational Data Model, 322
Assessors, 133, 135
Assistive technologies, for disabled users, 571–572
Associations, in class models, 312
Assumptions, validation of, 218–219, 504–505
Asynchronous processing, 500
ATAM (Architecture Tradeoff Analysis Method)
  architecture-centric activities, 226–229
  overview of, 222–223, 226
  stakeholder-centric activities, 229–230
Attack trees, 451–455
Attributes
  of architecture elements, 20
  in class models, 312
Audience
  clarity of AD presentation to, 203
  in TARA-style architectural review, 233
  targeting classes of, 35
  targeting in modeling, 188–189
Auditing
  ensuring accountability with, 444, 462
  as security mechanism, 445
  sensitive events, 458
Authentication
  as concern in Security perspective, 442
  as security mechanism, 445
  of system users, 459
Authorization
  of access, 459–460
  as concern in Security perspective, 442
  ensuring information secrecy, 460
  as security mechanism, 445
Authorized criteria, in stakeholder selection, 133
Availability
  as concern of Security perspective, 444
  protecting, 460–461
  as security mechanism, 445
Availability and Resilience perspective
  applicability to views, 512–513
  assessing against requirements, 524–525
  backup and disaster recovery solutions, 532–533
  capturing availability requirements, 516–517
  cascading failures and, 534–535
  clustering and load balancing for high-availability, 527–528
  concerns of, 512–516
  designing for failure, 530–531
  desired quality of, 437
  error detection as problem in, 536–537
  fault-tolerant hardware, 526–527
  fault-tolerant software, 530
  functional availability, 521–524
  incompatible technologies and, 539
  logging transactions, 528–529
  maintaining large information systems and, 50
  overambitious requirements as problem in, 536
  overestimating component resilience, 537–538
  overlooking global availability requirements, 538–539
  overview of, 511–512
  platform availability, 519–521
  producing availability schedule, 517–519
  relaxing transactional consistency, 532
  replicating components, 531
  single point of failure, 533
  software availability solutions, 529–530
  tactics for reworking architecture, 525–526
  unavailability due to overload, 535–536
Availability requirements
  assessing architecture against availability requirements, 524–525
  avoiding overambitious availability requirements, 536
  capturing availability requirements, 516–518, 539–540

B

Background, asynchronous processing in, 500
Backout strategy
  in installation model, 403–405

Backout strategy, *continued*
in migration model, 406
planning for, 419
Backup and restore
benefits of transaction logging, 528–529
as concern of Operational view, 399–401
risk of inadequate models for, 422–423
solutions for, 532–533
in third-party environments, 401
Bandwidth, 586–587, 589
BASE (Basically Available, Soft state, Eventual consistency), 306
Benchmark tests, 489
Big bang migration approach, 395
Black box approach, 146, 256
Boundary attributes, element, 20
Boxes-and-lines diagrams
for functional structure model, 276–278
for module structure model, 360
for network model, 384
for runtime platform model, 381
Braille display, for visually impaired users, 571
Brainstorming, in ATAM, 229
Build process
automating, 558
defining in codeline modeling, 367
reducing risk of lost environments, 563
Business analysts, 73
Business continuity, 516
Business drivers. *See also* Drivers
developing principles based on, 126
presenting in ATAM, 226
as problem-focused concern, 109–110
Business experts, 132
Business goals. *See also* Goals, 109–110
Business policies, as problem-focused concern, 111
Business principles, 126
Business standards, 111
Business state, 337
Business strategies, 105, 108

C

Candidate architectures
assessing, 64
defined, 18
internal organization and, 18–19
producing, 95
Capacity planning
for networks, 384
quantitative models and, 183
Cardinality, of entity relationships, 312
Cascading failure, avoiding, 534
Centralized systems
ensuring accountability, 462
ensuring information secrecy, 460
Certificates, user authentication and, 49
Change. *See also* Evolution perspective
create extensible interfaces, 553–554
data model change control, 302
driven by external factors, 547–548
facilitating with design techniques, 554–555
identify configuration strategy, 408–409
likelihood of change, 546
localizing effects of, 552–553
magnitude and dimensions of, 545–546
metamodel styles supporting, 555–556
preservation of knowledge during, 548
problems and pitfalls, 561
reliability of, 549, 558–559
timescale for, 547
when to pay for, 547
Character sets, internationalization, 581, 583
CIA (confidentiality, integrity and availability), 444
Clarity property, in effective AD, 203–204
Classes
in class models, 312
comparing views/viewpoints with objects/classes, 35
of incidents, 414–418
of services, 512–514, 520
of stakeholders, 133–138
Client/server structure, 16–18, 171
Clients
nodes of runtime platform, 378
reducing risk of unsecured, 470–471
Cloud computing, 401, 451
Clustering, for high-availability, 527–528
Coarse-grained operations, improving performance/scalability with, 502

Code, presenting AD in, 214
Code viewpoint, in Siemens set, 624
Codeline models
  activities, 366
  notation of, 366
  overview of, 365–366
Codeline, organization of, 359
Coherence, 254–255, 269
Cohesion, in functional design philosophy, 269
Collusion, insider threat and, 470
Commentary, on view model, 211
Committed criteria, in stakeholder selection, 133
Common design models
  activities, 365
  notation of, 363–365
  overview of, 362–363
Common processing
  defining, 362
  example, 363–364
  identifying, 358
  need for, 365
Communication
  models as tool for, 178, 256
  scenarios as tool for, 147
  skills of software architects, 77
  as stakeholder responsibilities, 141
  with stakeholders, 86
Communicators, classes of stakeholders, 133, 135
Compartmentalization, as security principle, 457
Compatibility
  internationalization and, 583
  of protocols in heterogenous networks, 587
  technology-related, 376, 539
Compensating transactions, information consistency and, 306
Complex systems
  breaking into interrelated views, 33–34
  limitations of applying monolithic models to, 32–33
Complexity
  evolution support adding to, 548
  excessive, 352–353
  using views/viewpoints to manage, 38
Component and Connector styles, in SEI viewpoint catalog, 625
Components
  architecture elements as, 21
  avoiding cascading failures, 534–535
  overreliance on specific hardware/software, 562
  replication of, 531
  resilience and, 537–538
Comprehensibility, qualities of good scenarios, 153
Computational viewpoint, in RM-ODP, 623
Conceptual viewpoint, in Siemens set, 624
Concerns
  capturing, 91
  driving architecture definition process, 86
  ISO documentation recommendations and, 206–207
  linking to principles and decisions, 125–128
  perspectives defining, 50, 57
  problem-focused, 107–111
  qualities of good concerns, 116
  real-world constraints as, 114–116
  relationship with requirements and architecture, 117
  scope of, 35
  separating/breaking down, 38, 270
  shaping architectural solutions, 105–106
  software architects considering wide range of, 66
  solution-focused, 107–114
  stakeholders and, 22
  summary and further reading, 128–129
  understanding/capturing, 68
  views addressing different, 45
Concerns, by perspective
  Accessibility perspective, 570–571
  Availability and Resilience perspective, 512–516
  Development Resource perspective, 575–576
  Evolution perspective, 545–549
  Internationalization perspective, 581–582
  Location perspective, 586–587
  Performance and Scalability perspective, 477–482

Concerns, by perspective, *continued*
- Regulation perspective, 592–593
- Security perspective, 442–446
- Usability perspective, 597–598

Concerns, by viewpoint
- Concurrency viewpoint, 335–339
- Context viewpoint, 248–255
- Deployment viewpoint, 374–377
- Development viewpoint, 358–360
- Functional viewpoint, 268–271
- Information viewpoint, 294–311
- Operational viewpoint, 394–402

Conciseness, properties of effective AD, 201–203

Concurrency viewpoint/views
- Accessibility perspective applied to, 569
- Availability and Resilience perspective applied to, 513
- checklist for, 355
- consistency across views, 431–434
- contention issues related to, 506
- defined, 245
- dependencies, 427
- Development Resource perspective applied to, 575
- Evolution perspective applied to, 544
- further reading, 355–356
- Internationalization perspective applied to, 580–581
- interprocess communication and, 336–337
- Location perspective applied to, 586
- mapping functional elements to tasks, 336
- overview of, 40–41, 333–335
- Performance and Scalability perspective applied to, 477
- Performance perspective applied to, 51
- problems and pitfalls, 351–355
- reentrancy, 338–339
- Regulation perspective applied to, 592
- scalability support, 338
- Security perspective applied to, 441
- stakeholder concerns, 339
- startup and shutdown, 338
- state management in, 337
- state models. *See* State models
- synchronization and integrity, 337
- system-level concurrency models. *See* System-level concurrency models
- system types and, 40
- task failure, 338
- task structure in, 335–336
- Usability perspective applied to, 596–597

Confidentiality, integrity and availability (CIA), 444

Configuration management
- build variation points into software, 557
- as concern in Operational viewpoint, 398
- defining in codeline modeling, 367
- reliable change via environment, 559
- reliable change with software management, 558

Configuration management models, 406–409

Connectivity
- Location perspective concerns, 586
- network connections, 383

Connectors
- designing, 283
- in functional structure model, 272

Consistency
- as concern of Context viewpoint, 254–255
- in functional design philosophy, 269
- of information, 305–306
- pitfalls related to view-and-viewpoint approach, 39

Consistency of views. *See* Views, consistency across

Constraints
- in AD, 25
- as concern of Development Resource perspective, 575–576
- design, 365
- in installation model, 403–404
- physical, 377
- real-world constraints as concerns, 114–116
- reducing, 421–422
- solutions shaped by, 105
- standards and policies as, 107

Construction
- evaluation techniques for system construction phase, 232–233
- incremental deliverables in, 88

Constructive characteristic, of good principles, 120
Consumers
  external interface as, 251
  separating information providers from information consumers, 309
Content equivalence technique, for disabled users, 572
Contention. *See* Resource contention
Context model
  activities, 258–260
  notation for, 257–258
  overview of, 255–256
Context viewpoint/views
  Accessibility perspective applied to, 569
  Availability and Resilience perspective applied to, 513
  completeness, consistency, and coherence, 254–255
  consistency across views, 427–428
  context model, 255–260
  defined, 244
  Development Resource perspective applied to, 575
  Evolution perspective applied to, 544
  external entities, services, and data in, 249–250
  external interfaces in, 251–252
  impact of system on its environment, 253–254
  interaction scenarios, 260–261
  interdependencies between entities, 252–253
  Internationalization perspective applied to, 580–581
  Location perspective applied to, 586
  overview of, 40–41, 247–248
  Performance and Scalability perspective applied to, 477
  problems and pitfalls, 261–264
  Regulation perspective applied to, 592
  Security perspective applied to, 441
  stakeholder concerns, 254–255
  system scope and responsibilities, 248–249
  system types and, 42
  Usability perspective applied to, 596–597
Controls
  administrative, 409–410
  data model change control, 302
  operational, 397
Conventions
  diagrams. *See* Semantics
  perspective creating, 57
Corporate assets, protecting, 593
Correctness
  checking technical correctness of architecture, 218
  properties of effective AD, 198–199
Costs
  of accommodating changes that do not happen, 560–561
  auditing, 460
  of change, 547
  as constraint, 115
  of deployment, 575–576
  formula for total operation cost, 491
  migration and, 395
COTS (custom off-the-shelf) package, 138
Coupling, in functional design philosophy, 270
Credibility, qualities of good scenarios, 153
Cross-cutting concerns, 6, 48
Cryptography
  avoiding ad hoc, 472
  confidentiality, 444
  information secrecy, 461
  integrity, 461–462
Cultural norms, internationalization and, 582
Currency conversion, internationalization and, 583
Currency, properties of effective AD, 204
Custom off-the-shelf (COTS) package, 138
Customers, as focus in modeling, 194

D

Data
  Context viewpoint/views and, 249–250
  identifying data entities, 314
  improving performance and, 502
  information systems and, 296
  interface design and, 283
  protecting, 593
  sharing, 341

Data flow model, 154
Data marts, star schema for, 312, 314
Data migration
  models for, 405–409
  operational concerns, 395–397
  operational problems and pitfalls, 420
  third-party environments and, 401
Data model
  change control, 302
  configuration management model as, 407
Data providers, 251
Data stores
  backup and restore planning, 401
  data migration concerns, 396
Data warehouses
  concurrency in, 334
  information systems and, 296
  star schema for, 312, 314
Databases
  configuration management, 398
  consistency of distributed, 327–328
  contention risk in, 506–507
  information storage models, 302–304
  locking, 173
  overloading, 327
DDoS (distributed denial-of-service) attacks, 461
De facto standards, 113
Deadlocks, 346, 353–354
Decision logs, 236
Decision points, evaluation tool for go/no go decisions, 219
Decisions
  architecturally significant, 124–125
  basing on technology principles, 126
  concerns influencing, 107
  deferring to "last responsible moment," 201
  design decisions in AD document, 209–210
  documenting, 211
  linking to principles and concerns, 125–128
  overview of, 122–124
  stakeholder responsibilities for making, 142
Decommissioning projects, 619–620
Decomposition
  applying to functional elements, 281
  structural, 314–315
Defense in depth principle, 457
Definitions
  conventions use in this book, 8
  including glossary in AD, 206
  of terms in models, 191
Deliverables
  creating executable, 194
  timeliness of, 200–201
Denial-of-service (DoS) attacks, 444, 461
Denormalizing data, 502
Dependencies
  analyzing, 558
  avoiding too many, 290–291
  clarity/accuracy of, 387–388
  configuration, 408
  identifying, 362, 403–404
  implicit dependencies missing, 262
  between views, 426–427
Dependency injection (Inversion of Control), 555
Deployment
  late consideration of environment for, 389–390
  rolling back unsuccessful, 559
Deployment viewpoint/views
  Accessibility perspective applied to, 570
  authenticating users and, 49
  Availability and Resilience perspective applied to, 513
  compatibility issues, 376
  consistency across views, 428–434
  defined, 42, 245
  dependencies between views, 427
  determining network capacity and requirements, 376–377
  Development Resource perspective applied to, 575
  Evolution perspective applied to, 545
  hardware availability in, 520
  intermodel relationships, 386–387
  Internationalization perspective applied to, 580–581
  Location perspective applied to, 56, 586

network models, 382–384
overview of, 373–374
Performance and Scalability perspective applied to, 477
performance-critical structures in, 485–486
physical constraints, 377
platform models, 378–382
problems and pitfalls, 387–391
reducing technology unavailability, 525–526
Regulation perspective applied to, 592
runtime platform required in, 374–375
Security perspective applied to, 442
specifying hardware and hosting requirements, 375
specifying third-party software requirements, 375–376
stakeholder concerns, 377
system types and, 40
technology dependency models, 384–386
Usability perspective applied to, 596–597

Descriptive naming, of models, 190

Design
assistive technologies for disabilities, 571–572
avoiding complexity in security, 457
boundary between architecture definition and, 67–68
creating set of design inputs, 87
decisions in AD document, 209–210
error handling standards in, 537
for failure, 530–531
functional design philosophy, 269–271
improving performance/scalability, 501–502
for security implementation, 453–455
separating design from requirements analysis, 65–66
Software architects making decisions regarding, 64
standard approach in common design models, 363–365
styles benefitting, 170
techniques facilitating change, 554–555

Design authorities, 73–75

Design patterns
applying, 172–174
architectural styles. *See* Styles
building variation points into software, 556–557
example of, 162–163
identifying and defining, 365
introduction to, 161–162
language idioms. *See* Language idioms
software design patterns. *See* Software design pattern
standardization with, 358
tactics and, 48, 166–167
techniques facilitating change, 554–555
using, 165–166

Detail
level in views, 35
too much, 367–368
wrong/inappropriate level of, 262–263, 289

Detection, security and, 444–445

Developers
classes of stakeholders, 133, 135–136
expanding focus to include all stakeholders not just developers, 2
software architect compared with, 75
using views/viewpoints to improve focus of, 38

Development. *See also* Evolution perspective
complexity concerns, 548
preservation of knowledge during change, 548
reducing risk of lost environments, 562–563
reliable change and, 560

Development Resource perspective
activities, 576–577
applicability to views, 574–575
concerns, 575–576
defined, 568
desired quality, 438
overview of, 573–574
problems and pitfalls, 577–578
tactics, 577

Development viewpoint, in “4+1” set (Kruchten), 622

Development viewpoint/views
  Accessibility perspective applied to, 569
  administration models, 411
  Availability and Resilience perspective applied to, 513
  codeline models, 365–367
  common design models, 362–365
  concerns, 358–360
  consistency across views, 428–433
  coordination between development/ operational staff, 419
  defined, 41, 245
  dependencies between, 427
  Development Resource perspective applied to, 575
  Evolution perspective applied to, 544
  Internationalization perspective applied to, 580–581
  Kruchten's standard views, 35
  Location perspective applied to, 586
  module structure models, 360–362
  overview of, 357–358
  Performance and Scalability perspective applied to, 477
  problems and pitfalls, 367–370
  Regulation perspective applied to, 592
  Security perspective applied to, 441
  system types and, 40
  Usability perspective applied to, 596–597
Diagrams. *See also* Notation
  conventions. *See* Semantics
  definitions in, 288
  for precision in presentation of information, 205
  in TARA-style architectural review of system, 233
Digital signatures, 460
Dimensional databases, 303
Disabled persons, regulations regarding. *See also* Usability perspective, 593
Disaster recovery. *See also* Backup and restore
  Availability and Resilience perspective, 515–516
  failure to specify, 391
  identifying solutions for, 532–533
  incident recovery analysis, 521–522
  location-related tactics, 588
Discovery, architecture definition as, 66
Disks
  archiving/retaining information on, 309
  availability and time to repair, 515
  backup and disaster recovery solutions, 532
  mirrored, 526–527, 532–533
Distributed databases, 327–328
Distributed denial-of-service (DDoS) attacks, 461
Distributed systems
  ensuring accountability with nonrepudiation, 462
  ensuring information secrecy, 460–461
Document Control section, in AD, 208
Document sign-off, 237
Documentation
  formal presentation of AD, 213
  wiki presentation of AD, 213–214
Domain architect, 72
Domain-specific languages (DSL), 186–187
Domains
  analysis of, 314
  TOGAF, 628–629
DoS (denial-of-service) attacks, 444, 461
Downtime, planned/unplanned, 514–515, 521–524
Drawing tools, for presentation of AD, 214
Drivers
  business drivers as problem-focused concern, 109–110
  shaping architectural solutions, 105
  technology drivers as solution-focused concern, 112
DSL (domain-specific languages), 186–187
Dynamic structure
  in airline reservation example, 16–17
  candidate architectures and, 19
  of a system, 13

E

EAI (Enterprise Application Integration), 169, 616
Elements
  assigning responsibilities to functional, 281–282

of common design models, 365
deciding what to include in a view, 35
defined, 20
of functional structure model, 271–272
identifying functional, 280–281
of network models, 382–383
overview of, 20–21
relationships between core concepts, 26–27
replaceability of, 556–557
of runtime platform models, 378–379
system elements and relationships, 12–13
Encapsulation, of change-related effects, 553
Engineering viewpoint, in RM-ODP, 623
Enterprise Application Integration (EAI), 169, 616
Enterprise architect, 73
Enterprise architecture frameworks
overview of, 627
TOGAF (The Open Group Architecture Framework), 628–629
Zachman framework, 627–628
Enterprise-owned information, 298–299
Enterprise resource planning (ERP), 138
Enterprise viewpoint, in RM-ODP, 623
Enterprise-wide services, 616–617
Entities
in entity-relationship modeling, 312
external. *See* External entities
identifying, 314
in life history models, 317
in state models, 347–348
Entity-relationship models, 187, 311–313
Environment
designing for deployment, 381
development problems related to, 369–370
impact of system on, 253–254
reduce risk of lost, 562–563
regulation concerns and, 593
reliable change with configuration management, 559
system quality scenario and, 151
ERP (enterprise resource planning), 138
Error conditions
administration models, 410–411, 413
detection of, 536–537
internationalization and, 583
Escalation process, in support model, 414–415, 418–419
ETL (Extraction, Transformation, and Load) tools, 396
Evaluation of architecture
applying to existing system, 233–236
choosing approach to, 237–238
formal reviews and structured walkthroughs, 220–222
overview of, 217–218
presentations for, 219–220
prototypes and proof-of-concept systems, 224–225
reasons for, 218–219
recording results of, 236–237
reports, 236
reworking and, 96
scenarios in, 222–223, 226–230
skeleton systems, 225
during software lifecycle, 230–233
techniques for, 219
Event entities, in state model, 347
Event providers, 251
Events, alert-related, 397–398
Eventual consistency approach, to information consistency, 306
Evolution perspective
achieving reliable change, 558–559
applicability to views, 51, 544–545
assessing ease of evolution, 551
characterizing evolution needs, 549–551
concerns in, 545–549
containing changes, 552–553
creating extensible interfaces, 553–554
design techniques facilitating change, 554–555
desired quality, 437
maintaining large information systems, 50
metamodel-based architectural styles, 555–556
overview of, 543–544
preserving development environments, 560
problems and pitfalls, 560–564
standard extension points, 557–558

Evolution perspective, *continued*
tradeoffs in, 552
variation points in software, 556–557
Execution coordination mechanisms, for interprocess communication, 341
Execution viewpoint, in Siemens set, 624
Expectations, managing, 577
Expertise, stakeholder, 35
Extensibility
creating extensible interfaces, 553–554
in functional design philosophy, 270
External checks, consistency across views, 426
External entities
in context model, 256
external interfaces and, 251–252
in functional structure model, 272
identifying, 249–250, 260
implicit dependencies and, 262
interdependencies between system and, 252–253
missing or incorrect, 261
nature and characteristics of, 250
overcomplicated interactions between, 264
trustworthiness of, 458
External hosting
insider threat and, 470
operational concerns, 401
security threats of, 450–453
Extraction, Transformation, and Load (ETL) tools, 396
Extreme Programming (XP), 101, 547, 607

F
Facilitation skills, of software architects, 76
Fact tables, in star schema, 312
Fail securely principle, 466
of Security perspective, 458
Failover, high-availability clustering, 527
Failure
avoid cascading, 533
avoid single points of, 533
design for, 530–531
Failure scenarios, 158
Fault-tolerance
hardware, 526–527
software, 530
Fault Trees technique, threat model, 451–452
Feature Driven Development, 100
File-based stores, 304
Finance, regulation concerns, 592
Finite state machine (FSM), 318
Finite State Processes language, 350
Fitness for purpose, development resources and, 577
Flexibility
analyzing, 284
of architectural decisions, 86
critical quality properties and, 560–561
of design patterns, 173
in functional design philosophy, 270
skills of software architects, 77
Flow diagrams, 415
Flow of information, 304
Focus, lack of or unevenness of, 368–369
Formal agreements, 219
Formal notations, 343
Fragmentation, view-and-viewpoint approach and, 39
FSM (finite state machine), 318
Full-scale live tests, 156–157
Functional availability, 521–525
Functional capabilities, 268
Functional cohesion, 553
Functional differences, internationalization and, 582
Functional elements
in functional structure model, 271–272
mapping to tasks, 336
Functional evolution, 546
Functional migration, 395
Functional requirements, 260
Functional scenarios
example of, 150–151
information in, 150
types of scenarios, 146
UML sequence diagram of, 154–155
Functional structure models
elements of, 271–272
non-UML notation, 276–280

types of qualitative models, 181
UML component diagrams, 273–275
Functional viewpoint/views
Accessibility perspective applied to, 569
assigning responsibilities to functional elements, 281–282
Availability and Resilience perspective applied to, 513
breaking AD document down by views, 205
checklist for, 291
comparing with Information, and Operational viewpoints, 46–47
concerns, 268–271
consistency across views, 427–431
defined, 41, 244
dependencies between, 427
designing connectors, 283
designing interfaces, 282–283
Development Resource perspective applied to, 575
Evolution perspective applied to, 51, 544
example of type of information in AD document, 210
functional structure model, 271–273
identifying functional elements, 280–281
Internationalization perspective applied to, 580–581
Location perspective applied to, 586
non-UML notation, 276–280
overloading of, 286–288
overview of, 267–268
Performance and Scalability perspective applied to, 477
problems and pitfalls, 285–291
reducing risk of concurrency-related contention, 506
Regulation perspective applied to, 592
Security perspective applied to, 441
system types and, 40
UML component diagrams, 273–275
Usability perspective applied to, 596–597
walkthroughs, traceability checks, and analysis, 284

G

Gane and Sarson information flow model, 316
Garland and Anthony viewpoint set, 626–627
Generalization
applying to functional elements, 281
in functional design philosophy, 270
patterns facilitating change, 555
styles, patterns, and idioms resulting in, 166
Global availability requirements, 538–539
Glossary, 206, 212
Goals
business goals as problem-focused concern, 109–110
performance/scalability and, 502–503
reviewing, 259–260
shaping architectural solutions, 105–106
in TARA-style architectural review of system, 233
technology goals as solution-focused concern, 112
"God object" problem, 290
Good enough approach, to modeling, 179, 194
Graphical notations. *See also* UML (unified modeling language)
of availability schedule, 518
for estimating functional availability, 523
for technology dependency model, 385
Groups
build variation points into software, 557
configuration, 407–408
identifying supported, 414–416
security policy defined by, 448
stakeholder, 141
Growth, as dimension of change, 546

H

Hardware
availability and time to repair, 515
degrade gracefully, 499–500
estimate platform availability, 519–521
fault-tolerant, 526–527
online/offline storage hardware, 378–379
overreliance on specific, 562
in platform evolution, 546
reducing compatibility risks, 539
resource requirements, 481

Hardware, *continued*
runtime platform activities related to, 381
scale up or scale out, 498–499
specifying type and quantity of, 375
virtualization tools, 563
Hash functions, cryptographic, 461–462
Headroom provision, in deployment, 390–391
Health and safety regulations, 593
High-availability, 527–528
High-contrast and low-resolution interfaces, for disabled persons, 571
Hosting requirements, specifying, 375
Hot spots, 506–507

I
IAF (Integrated Architecture Framework), 627
Identifiers, for information, 299–301
Identifying scenarios, 148–149
IDLs (Interface definition languages), 283
IEEE (Institute of Electrical and Electronics Engineers)
role defining open standards, 113
Standard 1471 (on views), 34
Improvements
perspective resulting in, 55
styles, patterns, and idioms resulting in, 166
In-house development, 615
Incident recovery analysis, 521–522
Informal notations, 343–344
Information
accountability, 462
consistency of, 305–306
disaster recovery of, 515–516
identifiers, 299–301
information, 307–308
integrity of, 461–462
ownership of, 296–298
privacy of, 445
purpose and usage of, 295–296
quality analysis, 320–321
quality of, 597
secrecy of, 460–461
storage models, 302–304
structure and content of, 294–295
synchronization in migration models, 406
timeliness, latency, and aging of, 308–309
Information capture skills, of software architects, 76
Information flows and ports, 277, 279–280
Information models
information flow models, 315–317
information lifecycle models, 317–319
information ownership models, 319–320
information quality analysis, 320–321
metamodels, 321–322
types of qualitative models, 181
volumetric models, 322
Information providers, separating from information consumers, 309
Information viewpoint, in RM-ODP, 623
Information viewpoint/views
Accessibility perspective applied to, 569
archiving/retaining information, 309–310
Availability and Resilience perspective applied to, 513
breaking AD document down by views, 205
comparing with Functional and Operational viewpoints, 46–47
concerns, 294–311
consistency across views, 427–433
consistency of information, 305–306
defined, 41, 244
dependencies between, 427
Development Resource perspective applied to, 575
enterprise-owned information, 298–299
Evolution perspective applied to, 544
flow of information, 304
identifiers for information, 299–301
information flow models, 315–317
information lifecycle models, 317–319
information ownership models, 319–320
information quality analysis, 320–321
Internationalization perspective applied to, 580–581
Location perspective applied to, 586
metamodels, 321–322
models, 311
overview of, 293–294
ownership of information, 296–298

Performance and Scalability perspective applied to, 477
problems and pitfalls, 322–329
purpose and usage of information, 295–296
quality of information, 307–308
Regulation perspective applied to, 592
Security perspective applied to, 51, 441
stakeholder concerns, 310–311
static information structure models, 311–315
storage models for information, 302–304
structure and content of information, 294–295
system types and, 40
timeliness, latency, and aging of information, 308–309
Usability perspective applied to, 596–597
user authentication and, 49
volatility of information semantics, 301–302
volumetric models, 322

Informed criteria, in stakeholder selection, 133
Infrastructure architect, 72
Inputs, to the architectural design process, 94
Input, provided by scenarios to architectural assessment, 147
Insider threat, 469–470
Installation groups, 403–404
Installation models, 402–405
Installation, operational concerns, 394
Institute of Electrical and Electronics Engineers (IEEE)
role defining open standards, 113
Standard 1471 (on views), 34

Instrumentation, 359
Integrated Architecture Framework (IAF), 627
Integration
of architectural decisions, 86
defining in codeline modeling, 367
evolution of, 546
lacking in production environment, 422

Integration hub, for interface complexity issues, 326
Integrity
defined, 461
identifying security policy requirements, 449
of information, 461–462
security concerns and, 444
as security mechanism, 445
security policy for, 448
synchronization of threads and, 337

Interaction scenarios, 260–261
Interactions, functional, 284
Interactive modeling, 194
Interface definition languages (IDLs), 283
Interfaces
assistive technologies for disabilities, 571–572
attributes of architecture elements, 20
complexity issues, 325–326
in context model, 256
designing functional, 282–283
extensibility of, 553–554
in functional structure model, 272
poorly defined, 285
usability concerns and, 597–598
usability tactics and, 599

Interfaces, external
functional concerns and, 268
identifying, 251, 260
loose or inaccurate, 262

Internationalization
in common design models, 364–365
design patterns for, 173

Internationalization perspective
activities, 582
applicability to views, 580–581
architectural tactics, 583
concerns, 581–582
defined, 568
desired quality, 438
overview of, 579–580
problems and pitfalls, 583

Internet chat technologies, for hearing impaired, 571
Internet, enabling project for, 618–619
Internet-scale systems
availability requirements for, 539
location concerns, 587
overloading, 535
relaxing transactional consistency in, 501

Interprocess communication. *See* IPC (interprocess communication)
Introduction and Management Summary section, in AD document, 209
Intrusion detection, 455
Inversion of Control (dependency injection), 555
IPC (interprocess communication)
  defining mechanisms for, 345–346
  overview of, 341
  types of mechanisms for, 336–337
ISO (International Organization for Standardization)
  financial services messaging (20022), 322
  metadata (11197-3), 321
  recommendations for documenting an architecture (42010), 206–207
  role defining open standards, 113
  sharing model across views and (42010), 58
IT strategies, 105, 112
Iterative approach
  reliable change with continuous iterations, 559
  to software development, 100
  to system delivery, 544

J
Jackson System Development, 276
Jargon, avoiding overuse of, 264

K
Kanji keyboards, 583
Key-matching problem, 324–325
Keys, security, 49
"Knee" in the performance curve, 482, 499–500
Knowledge, preserving of during change, 548

L
Language idioms
  applying, 172–174
  overview of, 165
  types of design patterns, 161
  using, 165–166
Languages. *See also* Internationalization perspective
  internationalization and, 582
  patterns creating common, 166
Large programs, 612–614
Latency
  estimating for networks, 384
  excessive, 328–329
  of information, 308–309
Law enforcement, 593
Layered Implementation style, 171
Layering patterns, to facilitate change, 555
Leadership, software architect role and, 70
Lean Software Development, 101
Least amount of privilege, security principle, 457
Legislation
  regarding disabilities, 568–570
  regulation concerns and, 592–593
  usability problems and, 572
Lifecycle
  agile methods in, 100–102
  architectural decisions in, 86
  architecture definition in, 98
  evaluation of architecture during, 230–233
  information lifecycle models, 317–319
  project lifecycle, 605
  state compared with, 337
Lightweight processes. *See* Threads
Likelihood of change, 546, 550–551
Links, network, 379
Lists, for presentation of information, 205
Live system, data migration from, 396–397
Load balancing, for high-availability, 527–528
Load, peak load behavior, 481–482
Local processing
  make design compromises, 502
  performance differences of remote vs., 504, 508–509
  performance model example, 487
Location perspective
  activities, 587–588
  applicability to views, 585–586
  applied to Deployment view, 56
  architectural tactics, 588–589

concerns, 586–587
defined, 568
desired quality, 438
overview of, 585
problems and pitfalls, 589
Locks, database, 173
Logical views, Kruchten's standard views, 35
Logs/logging
instrumentation and, 359
log transactions, 528–529
Lookup data, 296

M

Magnetic tape, for backup and disaster recovery, 532
Magnitude of change, 545, 550–551
Maintainers, classes of stakeholders, 133, 136
Management tools, 420–421
Master data, 296
Mathematics
mathematical model, 183
mathematical notation, 191
Mean time between failures (MTBF), 520
Mean time to repair (MTTR), 520
Measurement/measurability
business goals and drivers and, 109
estimate platform availability, 519–521
identify and estimate for performance models, 486
measure and estimate performance, 489
performance monitoring requiring, 399
perspective creating, 57
qualities of good concerns, 116
quantitative models and, 183
Meetings, minutes of, 236
Message bus, for interface complexity issues, 326
Message-oriented interactions, 277, 279–280
Messaging mechanisms, for interprocess communication, 341
Metamodels
architectural styles based on, 555–556
informational, 321–322
Metasystem approach to change, 547, 555–556
Metrics. *See* Measurement/measurability
Migration
data, 395–397
functional, 395
Migration models, 405–406
Minutes of meeting, 236
Mirrored disks
backup and disaster recovery solution, 532–533
fault-tolerance of, 526–527
Mock-ups, types of qualitative models, 181
Modeling languages
ADL (architecture description language), 184–185
DSL (domain-specific languages), 186–187
entity-relationship models, 187
for qualitative models, 182
UML (unified modeling language), 185–186
Models
abstraction in, 189
administration models, 409–414
agile approach to, 193–194
in architectural description, 25
availability, 519–521
avoiding overload using performance models, 536
codeline models, 365–367
common design models, 362–365
configuration management models, 406–409
context models, 255–260
creating performance models, 484–487
descriptive naming in and term definitions, 190–191
example of performance model, 487
importance of, 178–181
information flow models, 315–317
information lifecycle models, 317–319
information ownership models, 319–320
information quality analysis, 320–321
installation models, 402–405
interaction scenarios in, 260–261
intermodel relationships, 386–387
metamodels, 321–322
migration models, 405–406
module structure models, 360–362

Models, *continued*
  network models, 382–384
  notation in, 191–192
  overview of, 177–178
  performance analysis, 487–488
  purposeful approach to creating, 187–188
  qualitative, 181–182
  quantitative, 182–183
  reducing risk of unrealistic performance, 503
  reworking architecture to improve performance, 490
  risk-driven approaching to, 189–190
  runtime platform models, 378–382
  semantics in, 192
  simplicity in, 191
  sketches, 184
  state models, 347–351
  static information structure models, 311–315
  support models, 414–419
  system-level concurrency models, 340–347
  targeting an audience with, 188–189
  technology dependency models, 384–387
  updating, 193
  validation of, 193
  for views, 210–211
  volumetric models, 322
  for well-defined security, 468–469
Moderator role, in reviews and walkthroughs, 221
Module structure models
  activities, 362
  notation of, 360–361
  overview of, 360
Module styles, in SEI viewpoint catalog, 625
Module viewpoint, in Siemens set, 624
Modules, 21, 358
Monitoring
  administration models for, 409–410
  concerns in third-party environments, 401
  operational control and, 397
  performance, 399
  as security mechanism, 445
Moore's Law, 476
MTBF (Mean time between failures), 520
MTTR (Mean time to repair), 520

N
Naming, descriptive naming of models, 190
Negotiation skills, of software architects, 76
Network models
  activities of, 384
  elements of, 382–383
  example of, 387
  notation of, 384
Networks
  capacity needs of, 376
  connections, 383
  designing, 384
  failure of, 588
  hardware requirements for, 376
  links in, 379
Nodes
  clustered configurations and, 527
  in network models, 378, 382–383
Nonfunctional requirements, issues addressed by perspectives, 48
Nonrepudiation of messages, 445, 462
Normalization, of information models, 314
NoSQL databases, 303–304
Notation
  administration models, 411
  codeline models, 366
  common design models, 363–365
  configuration management models, 407
  context models, 257–258
  information flow models, 316–317
  information lifecycle models, 318
  installation models, 403
  interaction scenarios, 261
  migration models, 405–406
  in models generally, 191–192
  module structure models, 360–361
  network models, 384
  non-UML notation, 276–280
  overview of, 273
  performance models, 485
  perspective creating, 57
  runtime platform models, 379–381
  state models, 348–350
  static information structure models, 312–314
  system-level concurrency models, 341–344

technology dependency models, 385–386
UML component diagrams, 273–275
Notifications, alert-related, 397–398
Numbering element of AD, 205

O

Object Constraint Language (OCL), 283
Object ID, 299
Object Modeling Technique, 276
Object-orientation
comparing views/viewpoints with objects/ classes, 35
object ID in, 299
OCL (Object Constraint Language), 283
Off-the-shelf deployment project, 138–139
Office space, development concerns, 576
Offline modes, 588
OLAP (online analytical processing), 296
OLTP (online transactional processing), 295
Online backups, 533
Open standards, 113
Operating systems, configuration management and, 398
Operational constraints, 115
Operational monitoring, 397, 410
Operational service levels, 516–517
Operational viewpoint/views
Accessibility perspective applied to, 570
administration models, 409–414
alerting, 397–398
Availability and Resilience perspective applied to, 513
backup and restore, 399–401
comparing with Functional and Information viewpoints, 46–47
configuration management, 398
configuration management models, 406–409
consistency across views, 431–434
data migration, 395–397
defined, 42, 245
dependencies between, 427
design functional availability schedule, 523–524
Development Resource perspective applied to, 575
Evolution perspective applied to, 545
functional migration, 395
installation and upgrade, 394
installation models, 402–405
Internationalization perspective applied to, 580–581
Location perspective applied to, 586
migration models, 405–406
operational monitoring and control, 397
overview of, 393–394
Performance and Scalability perspective applied to, 477
performance monitoring, 399
problems and pitfalls, 419–423
Regulation perspective applied to, 592
Security perspective applied to, 442
stakeholders, 401–402
support concerns, 399
support models, 414–419
system types and, 40
third-party environment and, 401
Usability perspective applied to, 596–597
user authentication and, 49
Optimization
consolidating related workload, 494–495
repeated processing and, 491–492
Organizational context, software architect role in, 73–75
Organizational or cultural constraints, 116
Organizational standards, 113
Overhead, transaction, 494–495
Overloading
availability and, 535
of central database, 327
degrade gracefully and, 499–500
functional, 286–288
Overview statement
in functional scenario, 150
in system quality scenario, 151
Ownership
of architecture definition, 68
of information, 296–298

P

Packages, implementing, 618
Paper models, 154–155
Parallel processing, 497–498
Parallel run migration approach, 395

Partitioning
performance and scalability and, 497–498
reduce risk of inappropriate, 504
Partnered development project, 140
Patterns. *See* Design patterns *and* Software patterns
Peak load behavior
improving, 495–496
performance and scalability and, 481–482
Percentages, availability metrics, 519
Performance
data migration concerns, 395
usability concerns, 598
Performance and Scalability perspective
analyzing performance models, 487–488
applicability to views, 476–477
applied to Concurrency viewpoint, 51
assessing performance against requirements, 489–490
asynchronous processing, 500
capturing performance requirements, 482–484
conducting practical tests, 488–489
consolidating related workloads, 494–495
creating performance models, 484–487
degrading gracefully, 499–500
design compromises, 501–502
desired quality, 437
distributing processing over time, 495–496
example applying, 55
maintaining large information systems, 49
minimizing resource sharing, 496
optimizing repeated processing, 491–492
overview of, 475–476
partitioning and parallelizing, 497–498
prioritizing processing, 493–494
problems/pitfalls, 502–509
reducing contention, 492–493
relaxing transactional consistency, 501
reusing resources and results, 496–497
reworking architecture to improve performance, 490
scaling up or out, 498–499
Performance-Critical structures, 485–486
Performance engineering, 399
Performance models
analyzing, 487–488
avoiding overloading, 536
creating, 484–487
example of, 487
reducing risk of unrealistic performance, 503
reworking architecture to improve performance, 490
Performance monitoring
administration models, 410–414
operational concerns and, 399
operational monitoring vs., 410
Persistent storage, 515
Perspectives
Accessibility. *See* Accessibility perspective
applying to models, 178
applying to views, 51–54
Availability and Resilience. *See* Availability and Resilience perspective
benefits of, 56–58
catalog of, 60–61, 437–438
comparing with viewpoints and views, 59
consequences of applying, 54–56
defined, 6, 47
Development Resource. *See* Development Resource perspective
Evolution. *See* Evolution perspective
Internationalization. *See* Internationalization perspective
Location. *See* Location perspective
overview of, 48–51, 567–568
Performance and Scalability. *See* Performance and Scalability perspective
pitfalls related to, 58
presenting for views, 210–211
quality properties and, 45–47
Regulation. *See* Regulation perspective
relationships between core concepts, 56
Security. *See* Security perspective
in software architecture example, 6
system types and, 61
Usability. *See* Usability perspective
viewpoints compared with, 58–60
views compared with, 45
Petri Nets, 350
Physical constraints
real-world constraints as concerns, 115
taking into account, 377

Physical environment, 587, 589
Physical sites, ignoring intersite complexities, 389–390
Physical viewpoint, in "4+1" set (Kruchten), 622
Physical views, Kruchten's standard views, 35
Piloting, development resources and, 577
Pipes and Filters architectural style, 167–169
Plan-driven projects, 609–611
Planned downtime, 514–515
Platform
- assumptions, 504–505
- evolution, 546

Platform availability
- assess against availability requirements, 524–525
- create incident recovery analysis, 521–522
- estimate, 519–521
- reduce risk of incompatible technologies, 539
- select fault-tolerant hardware, 526–527

Policies
- business policies as problem-focused concern, 111
- security. *See* Security policies
- shaping architectural solutions, 105
- technology policies as solution-focused concern, 113–114

Politics
- high-priority stakeholders, 132
- internationalization and, 582

Ports and information flows, 277, 279–280
Power grids, cascading failure of, 534
Practical testing
- performance and scalability and, 488–489
- reducing risk of unrealistic performance, 503
- simulating runtime environment in, 503

Precision
- lack of, 369
- properties of effective AD, 205
- qualities of good scenarios, 153
- of security policy, 448

Predictability, performance and scalability and, 480–481
Presentations
- of AD, 213–215
- for evaluation of architecture, 219–220
- for scope and option explorations, 231

Presenter role, 221
Primary keys, 299
Principals
- authentication of, 459
- authorize access for, 459–460
- granting least amount of privilege possible to, 457
- grouping for security policy, 448–450
- security, 440
- Security perspective concerns, 442

Principles
- applying recognized, 456–457
- conventions use in this book, 8
- creating own, 122
- definition of, 119
- examples of use of, 118–120
- general architectural principles in AD document, 209
- linking to concerns and decisions, 125–128
- overview of, 117–119
- qualities of good principles, 120–121
- for translating goals into features, 110
- view-specific, 210

Prioritization
- of evolutionary dimensions, 560–561
- of processing, 493–494
- of scenarios, 149, 229–230

Privacy, of information, 445
Privilege, principle of least, 457
Problem escalation, 401
Problem-focused concerns, 107–111
Procedure call mechanisms
- for interprocess communication, 341
- modeling using UML, 342

Process groups, in system-level concurrency model, 340
Process viewpoint, in "4+1" set (Kruchten), 622
Process views, Kruchten's standard views, 35

Processes
 context in architecture definition, 87–89
 exit criteria in architecture definition, 97–98
 flow of, 597
 interprocess communication, 336–337, 341
 outcomes in architecture definition, 86–87
 prioritizing, 346
 in system-level concurrency model, 340
 tasks and, 335
Processing
 areas of common processing, 358
 asynchronous, 500
 build variation points into software, 557
 consolidating related workloads, 494–495
 distributing over time, 495–496
 minimizing resource sharing, 496
 optimizing repeated, 491–492
 partitioning and parallelizing, 497–498
 prioritizing, 493–494
 reusing resources and results, 496–497
Processing nodes
 in network models, 382
 of runtime platform, 378
Processing pipeline, 168
Product architect, 72
Product management, 545
Product ownership, 545
Production engineers, classes of stakeholders, 133, 136
Production environment
 reducing constraints in, 421
 reducing lack of integration in, 422
Products
 in architectural description, 24–25
 development projects, 615–616
Program code, 191
Programming languages, 173–174, 282–283
Project lifecycle, 605
Project managers, 73
Projects
 agile, 607–609
 decommissioning, 619–620
 enabling for Internet, 618–619
 for enterprise-wide services, 616–617
 for extending existing systems, 617–618
 in-house development of, 615
 implementing software packages, 618
 large programs, 612–614
 for new product development, 615–616
 plan-driven, 609–611
 project lifecycle and, 605
 small and low-risk, 606–607
Proof-of-concept systems, 224–225
Proprietary standards, 113
Prototype implementation tests, 156
Prototypes
 for architectural definition phase of lifecycle, 232
 for defining scope and exploring options, 231
 of development resources, 577
 in evaluation of architecture, 224–225
 types of qualitative models, 181
Proxy stakeholders, 140–141
Publisher/Subscriber style, 171

Q

Qualitative models, 181–182
Quality attribute tree, in ATAM, 228
Quality management standards, 86
Quality, of information
 analyzing, 320–321
 overview of, 307–308
 poor quality information, 328
Quality properties
 in airline reservation example, 15–18
 internal organization and, 19
 modules and, 362
 perspective pitfalls and, 58
 perspectives and, 45–47
 scale of, 202
 in software architecture design, 5–6
 summarizing in AD document, 211
 types of system properties, 14
Quality triangle, 23
Quantifiability
 of business goals and drivers, 109
 of concerns, 116
Quantitative goals, 482–484
Quantitative models, 182–183

R

Race conditions, 354–355

RAID (Redundant Array of Inexpensive or Independent Disks) architectures, 526–527, 538
Rational Unified Process. *See* RUP (Rational Unified Process)
Recommendations, in TARA-style architectural review of system, 235
Recovery, from disaster. *See* Disaster recovery
Recovery, security detection and, 444–445
Redundant Array of Inexpensive or Independent Disks (RAID) architectures, 526–527, 538
Reentrancy, concurrency and, 338–339
Reference data, in information systems, 296
Regulation perspective
  activities, 594
  applicability to views, 591–592
  architectural tactics, 594
  concerns, 592–594
  defined, 568
  desired quality, 438
  in maintaining large information systems, 50
  overview of, 591
  problems and pitfalls, 594
Regulations
  disability requirements, 570
  usability and, 572
Relational databases, 302–303
Relationships, in entity-relationship modeling, 312
Release process
  ad hoc management and, 563–564
  automating, 559
  defining in codeline modeling, 367
Reliability
  of change, 549, 558–559
  usability and, 598
Remote procedure calls, 342
Remote processing, 504, 508–509
Repeated processing, optimize, 491–492
Replication
  applying to functional elements, 281
  component, 531
  reducing contention via, 492–493
Reporting database, for information systems, 295
Representative criteria, in stakeholder selection, 133
Request handling, overloading and, 535
Requirements
  assessing architecture against availability requirements, 524–525
  avoiding overambitious availability requirements, 536
  capturing availability requirements, 516–518, 539–540
  converting goals and drivers into, 109
  development resources, 578
  evolution of, 550–551
  identifying and prioritizing scenarios, 148
  location, 590
  performance and scalability and, 482–484, 489–490
  as problem-focused concern, 110–111
  process outcomes, 86
  regulation, 594–595
  relationship with concerns and architecture, 117
  revisiting, 96
  scenarios for capturing, 145–146
  scenarios for finding missing, 147
  security, 468–469, 572
  TARA-style architectural review of system, 234–235
  usability, 570, 600
Requirements analysis
  boundary between architecture definition and, 66
  as context for architecture definition, 88
  separating from design, 65–66
Resilience. *See* Availability and Resilience perspective
Resource contention
  analyzing, 346
  concurrency-related, 352–353, 505–506
  improving performance by reducing, 492–493
  minimizing resource sharing, 496
  reducing risk of, 505–507
Resources
  authorize access to, 459–460
  careless allocation of, 508

Resources, *continued*
constraints causing software projects/ delays, 577–578
at core of system security, 440
defining mechanisms for sharing, 345
designing security for sensitive resources, 453–455
development resources, 574–579
ensuring information secrecy, 460–461
identifying for security policy, 446–449
identifying threats to, 453
minimizing sharing, 496
reusing, 496–497
security concerns and, 442
Response time
defined, 477
hardware resources effecting, 481
interrelationship with throughput, 479–480
peak load behavior and, 482
performance and scalability concerns and, 477–479
Performance and Scalability perspective, 477–479
specifying requirements for, 484
Responsibilities
assigning to functional elements, 281–282
attributes of architecture elements, 20
context viewpoint concerns, 248–249
of external entities, 260
poorly understood, 285
of software architects, 77–78
of stakeholders, 141–142
Responsiveness class, response time, 478–479
Restore. *See* Backup and restore
Reuse, of resources and results, 496–497
Review records, 236
Reviewers
in architecture definition, 98
in reviews and walkthroughs, 221
stakeholder responsibilities, 142
Reviews
for architectural definition phase of lifecycle, 232
for defining scope and exploring options, 231
formal reviews for evaluating architecture, 220–222
for system construction phase of lifecycle, 232–233
Risk-driven approach, 189–190
Risks
assessing development resources, 576
assessing ease of evolution, 551
assessing performance, 490
due to unfamiliar technology, 202
functional migration and, 395
identifying availability, 525
operations and, 419–423
reducing, 166
risk assessment process, 455–456
RM-ODP (Reference Model for Open Distributed Processing), 623
Roadmaps, in business strategy, 108
Routine operational procedures, administration models, 410–412
Runtime containers, 378
Runtime dependencies, 386
Runtime platform, 374–375
Runtime platform models
activities of, 381–382
elements for, 378–379
notation of, 379–381
RUP (Rational Unified Process)
in development resources, 577
iterative approaches to software development, 100
Kruchten's approach as basis of, 34
plan-driven methods, 609

S

SAAM (Software Architecture Assessment Method). *See also* Scenario-based evaluation, 223, 226
Safety regulations, 593
Sarbanes-Oxley Act, 592
Scalability. *See also* Performance and Scalability perspective
concerns, 480
concurrency and, 338
scaling up or out hardware, 498–499
specifying requirements, 484
Scenario-based evaluation

in architectural definition phase of lifecycle, 232
architecture-centric activities, 226–229
overview of, 226
stakeholder-centric activities, 229–230
steps in, 222–223

Scenarios
in AD document, 211
applying, 154
capturing, 149–153
checklist for, 159
documenting, 211
effective use of, 157–159
in evaluation of architecture, 222–223, 226–230
identifying, 94, 148–149
overview of, 145–146
paper models for, 154–155
prioritizing, 148–149, 229–230
qualities of good scenarios, 153–154
simulations of, 156
testing, 156–157
types of, 146
uses for, 147–148
walkthroughs, 155–156, 260, 284

Schedule, availability, 517–519, 522–524

Scope
deciding what to include in a view (view scope), 34–35
defining in architectural description, 25
defining initial, 91
scenarios in validation of system scope, 147
of stakeholder concerns, 35
system scope as concern, 110–111
techniques for defining, 230–231

Scope creep, 263

Screen magnifier, for visually impaired users, 571

Screen reader, for visually impaired users, 571

Scrum, 101, 607

SDL, 350

Secrecy of information, 460–461

Security
data migration concerns, 395
defined, 440

Security infrastructure
assess risks, 455–445
avoid system not designed for failure, 466
design system-wide, 453–455
use third-party, 464–465

Security mechanisms
enforcing policies, 440
Security perspective concerns, 445–446

Security perspective
applicability to views, 441–442
applied to Information viewpoint, 51
concerns, 442–446
desired quality, 437
example of applying, 55
maintaining large information systems, 49
overview of, 439–441

Security perspective activities
assessing risks, 455–456
defining security policy, 448–450
designing security implementation, 453–455
identifying sensitive resources, 446–448
identifying threats, 450–453

Security perspective problems/pitfalls
ad hoc security technology, 472
assuming client is secure, 470–471
complex policies, 465
failure to consider time sources, 467–468
ignoring insider threat, 469–470
lack of administration facilities, 466–467
no clear requirements or models, 468–469
overreliance on security technology, 468
piecemeal security, 472
security as afterthought, 469
security embedded in application code, 471–472
system not designed for failure, 466
technology-driven approach, 467
unproven technologies, 465–466

Security perspective tactics
applying recognized security principles, 456–459
authenticating principals, 459
authorizing access, 459–460
ensuring accountability, 462
ensuring information integrity, 461–462

Security perspective tactics, *continued*
ensuring information secrecy, 460–461
integrating security technologies, 463
protecting availability, 462–463
providing security administration, 464
third-party infrastructure, 464–465
Security policies
avoiding complex, 465
concerns, 442–443
defining, 448–450
designing detection and recovery approach, 455
ensuring well-defined security models and requirements, 469
providing administration of, 464
resource address, 440
SEI (Software Engineering Institute) "Views and Beyond" Views, 624–625
Semantics
careful use of implied semantics in models, 192
representation incompatibilities, 322–324
volatility of information semantics, 301–302
Sensitivity points, in ATAM, 229
Separate responsibilities, security principle of, 457
Service-level agreements (SLAs), 388–389
Service providers, 251
Services
capturing availability requirements, 516–517
classes of, 512–514
enterprise-wide service projects, 616–617
provided by architecture elements, 20
Shared resources, 496
Sharing information, 194
Shutdown, concurrency design and, 338
Siemens viewpoint set, 623–624
Signatures, cryptographic, 444
Simplicity
in functional design philosophy, 270
in models, 191
Simulations
of scenarios, 156
types of qualitative models, 181
Single point of definition, localizing effects of change, 553
Single points of failure, 533
Skeleton systems
for architectural definition phase of lifecycle, 232–233
creating, 92
in evaluation of architecture, 225
Sketches
functional, 277
of functional and deployment views, 234
for informal modeling, 184
Skills
of model builder, 179
real-world constraints as concerns, 115
of software architect role, 76–77
SLAs (service-level agreements), 388–389
Small projects, 606–607
Software
applying availability solutions, 529–530
availability and time to repair, 515
build variation points for system evolution, 556–557
estimating platform availability, 519–521
fault-tolerant, 530
overreliance on specific, 562
in platform evolution, 546
reducing risk of incompatible technologies, 539
reliable change and, 558
selecting in common design models, 363, 365
third-party software requirements, 375–376
Software architects
in architectural description (AD), 64
architectural leadership, 70
aspects of, 68
boundary between AD and requirements analysis, 66
boundary between architecture definition and design, 67–68
involvement during stages of system delivery, 69–70
in organizational context, 73–75
overview of, 63
project lifecycle and, 605
relationships between core concepts, 71–72

responsibilities of, 77–78
separating design from requirements analysis, 65–66
skills of, 76–77
specialization areas for, 72–73
Software architecture
agile projects and, 607–609
applying metamodel-based styles, 555–556
approaches in ATAM, 228
assessing current ease of evolution, 551
considering evolution tradeoffs, 552
core concepts, 26–27
defined, 11–12
Development Resource perspective and, 574–579
evaluating. *See* Evaluation of architecture
fundamental system properties, 13–14
importance of, 19–20
ISO recommendations for documenting, 206–207
key activities, 84
in large programs, 612–614
overview of, 11–12
in plan-driven projects, 609–611
presenting in ATAM, 227
principles of design and evolution, 14–15
project lifecycle and, 605
refining, 64
relationship with requirements and concerns, 117
revising for evolution strategy, 552
reworking to improve performance, 490
scenarios in evaluation of, 147
in small and low-risk projects, 606–607
structures resulting from design decisions, 64
system elements and relationships, 12–13
system properties and internal organization, 15–19
tactics for reworking availability, 525–526
usability concerns, 598
Software design patterns
building patterns (Alexander) and, 161
example of use of, 162–163
overview of, 165
Software development lifecycle
Agile methods in, 100–102
evaluation of architecture during, 230–233
iterative approaches to, 100
overview of, 98
waterfall approaches to, 99–100
Software engineering practices, 86
Software packages, implementing, 618
Software product development project, 139–140
Solution architect, 72
Solution-focused concerns, 112–114
Solutions
concerns shaping, 105–106
criticality of problems and, 202
focusing on in modeling, 194
identifying and evaluating, 87
perspectives providing for common problems, 58
styles for finding related solutions, 170
Source code
codeline organization, 359
designing structure for, 367
Specialists
high-priority stakeholders, 132
lack of (Deployment viewpoint pitfalls), 389
Specializations, for software architect role, 72–73
Specificity, qualities of good scenarios, 153
Spreadsheets, 215
SQL databases, 302
SSADM data flow model, 316
Staged migration approach, 395
Stakeholders
applying Usability perspective, 571
approving security policy, 448
avoiding overambitious availability requirements, 539
capturing needs of, 64
clarity of, 203
classes of, 133–138
communicating with, 38
concurrency concerns, 339
context viewpoint concerns, 254–255
correctness in representing needs/ concerns of, 199

Stakeholders, *continued*
criteria for good, 133
defined, 6, 21, 131
deployment concerns, 377
determining audience class(es) view is aimed at, 35
development concerns, 359–360
development resource concerns, 576
engaging, 91
evaluating AD with, 220–222
evaluating architecture with, 96
functional concerns, 271
groups, 141
high-priority, 132
identifying and engaging, 68
identifying and prioritizing scenarios, 148
importance of, 23–24
individual, team, or organization, 22
information concerns, 310–311
interests and concerns of, 22–23
involving in scenarios, 158–159
involving in security administration, 464
ISO recommendations for documenting an architecture, 206–207
managing expectations of, 86–87, 110
in off-the-shelf deployment project example, 138–139
operational concerns, 397, 401–402
overview of, 21
in partnered development project example, 140
perspective pitfalls and, 58
presenting complex systems to, 33
proxy stakeholders, 140–141
reconciling needs of multiple, 288–289
relationships between core concepts, 26–27
responsibilities of, 141–142
scenarios for communication with, 147
security examples for, 469
selecting, 131–133
software architects getting input from, 66
in software architecture example, 2–4
in software product development project, 139–140
support models for, 414–419
Stakeholders section, in AD document, 209
Standard extension points, Evolution perspective, 557–558
Standardization
of design, 358–359
styles, patterns, and idioms as aid to, 166
of testing, 358–359
Standards
for alerts, 398
business standards as problem-focused concern, 111
disability requirements, 570
shaping architectural solutions, 105
technology standards as solution-focused concern, 113–114
Star schema (multidimensional schema or cube), for modeling data warehouses and data marts, 312, 314
Startup, concurrency design and, 338
State entity, 347
State machine, 347
State management
Concurrency view for, 337
designing state transitions, 350–351
identifying states, 350–351
State models
activities, 350–351
entities in, 347–348
notation of, 348–350
State transition models (state charts), 317–318, 348–350
Static data, in information systems, 296
Static information structure models
activities in, 314–315
notation of, 312–314
overview of, 311–312
Static structures
in airline reservation example, 16–17
candidate architectures and, 19
of a system, 12–13
Statistics tracking service, 534
Storage hardware, 378
Storage models, 302–304
Store of knowledge
perspective as, 50
styles, patterns, and idioms for, 165

Strategies
business strategies as problem-focused concern, 108
conventions use in this book, 8
IT strategies as solution-focused concern, 112
migration models, 406
shaping architectural solutions, 105
Structural decomposition. *See* Decomposition
Structure, internal, 268–269
Structure of information, 294–295
Styles
architectural description (AD) and, 172
benefits of, 170–171
checklist for, 174
defined, 164
example of use of, 167–169
identifying, 95
overview of, 164
in SEI viewpoint catalog, 624–625
two-tier client server approach, 16
types of design patterns, 161
using, 165–166
Sufficiency, properties of effective AD, 199–200
Suppliers, classes of stakeholders, 134, 136–137
Support models, 414–419
Support, operational concerns, 399
Support providers, 414–417
Support staff, classes of stakeholders, 134, 137
Symbolic notation, 191
Synchronization, integrity and, 337
SysML functional model, 279
System
architecture of, 20
design using styles, 170
elements and relationships, 12–13
fundamental properties, 13–14
impact on its environment, 253–254
projects for extending existing, 617–618
properties for internal organizations, 15–19
relationships between core concepts, 26
required behavior, 151
response required in functional scenario, 150
state, 150–151
System administrators
classes of stakeholders, 134, 137
as customers of administration models, 409–413
performance monitoring by, 399
System availability model, 519–521
System-level concurrency models
activities, 344–347
items in, 340–341
notation of, 341–344
System operations
defining sensitive areas in security policy, 449
optimizing repeated processing, 491–492
System quality, in TARA-style architectural review of system, 234–235
System quality scenarios
benefits of, 158
example, 151–152
information in, 151
types of scenarios, 146
System scope
context viewpoint concerns, 248–249
implicit or assumed, 264
as problem-focused concern, 110–111
validating, 147
Systems and Software Engineering-Recommended Practice for Architectural Description of Software-Intensive Systems (ISO 42010), 206–207

T
Table of Contents, in AD document, 208
Tables, for precision in presentation of information, 205
Tactics
for dealing with business goals and drivers, 109–110
defined, 48
design patterns and, 166–167
structuring perspective definition by, 51
TARA (Tiny Architectural Review Approach), 233–236
Tasks, in Concurrency viewpoint
failure of, 338
mapping functional elements to, 336, 344
structure of, 335–336

Team Software Process, 609
Teams
agile, 607–608
for modeling, 193–194
of stakeholders, 22
Technical constraints, 115
Technical evaluation
conducting for runtime platform model, 382
conducting for technology dependency model, 386
Technical integrity, 219
Technical knowledge, 389
Technical state, 337
Technologies
assistive, for disabled users, 571–572
avoiding overambitious availability requirements, 539
compatibility issues, 376
development resource concerns, 575
identifying/validating environment and platform assumptions, 504–505
increasing availability, 525–526
reducing risk of incompatible, 539
risks due to unfamiliar technology, 202
technology agnostic architectural decisions, 86
unproven, 388
Technologies, security
assessing, 455
avoiding ad hoc, 472
avoiding embedding in application code, 471–472
avoiding overreliance on, 468
avoiding technology-driven approach, 467
avoiding unproven, 465
integrating, 455, 463
providing administration of, 464
Technology dependency models
activities of, 386
notation of, 385–386
overview of, 384–385
Technology drivers, as solution-focused concern, 112
Technology experts, 132
Technology goals, as solution-focused concern, 112
Technology leadership role, of architects, 70
Technology policies, as solution-focused concern, 113–114
Technology principles, developing from business principles, 126
Technology specialists, software architect compared with, 75
Technology standards, as solution-focused concern, 113–114
Technology viewpoint, in RM-ODP, 623
Terminology, defining terms and symbols in models, 191
Testers, classes of stakeholders, 134, 137
Tests/testing
automated, 559
avoiding unavailability through overload, 536
component resilience, 538
conducting practical, 488–489
driven by scenarios, 147
full-scale live tests, 156–157
prototype implementation tests, 156
scenarios, 156–157
testability of good principles, 120
testability quality of good concerns, 116
Text and tables
administration models in, 411
assessing availability requirements, 524–525
availability requirements in, 516
availability schedule, 518
characterizing evolution needs, 551
codeline models in, 366–367
configuration management model in, 407
functional availability, 523
identify sensitive resources, 447
installation model in, 403
migration model in, 405
performance model in, 484–485
platform availability, 519
runtime platform model in, 381
for security policy, 448–450
support models in, 415
technology dependency model in, 386
threat model in, 451
Text-based approach
assessing current ease of evolution, 551

considering evolution tradeoffs, 552
presenting internationalization concerns, 581
Third-party environments
avoiding overambitious availability requirements, 539
operational concerns, 401
raising and monitoring alerts, 398
reducing risk of incompatible technologies, 539
security threats of system hosted in, 451
untrusted until proven otherwise, 458
using third-party security infrastructure, 464–465
Third-party software requirements, 375–376
Threads
determining threading design, 345
prioritizing, 346
in system-level concurrency model, 340
tasks and, 335
in thread-based concurrency model, 344
Threat model
avoid overreliance on technology, 468
ensuring well-defined security models and requirements, 469
Security perspective and, 450–453
using minimum amount of cryptography, 461
Threats
assessing, 455–456
designing mitigation features, 453–455
insider, 469–470
protecting availability, 462–463
security concerns and, 442–443
Three Peaks model, 87–88
Three-tier client server approach, 18
Throughput
defined, 479
effect of hardware resources on, 481
Performance and Scalability perspective, 479–480
specifying requirements for, 484
Tightly coupled design, 502
Time, real-world constraints as concerns, 115
Time sources, 467–468
Time to repair, Availability and Resilience perspective, 515
Time zones, 587, 589
Timeliness
of information, 308–309
properties of effective AD, 200–201
Timeouts for service calls, 499–500
Timescale for change, 547, 550–551
Tiny Architectural Review Approach (TARA), 233–236
Touch points, for usability, 571
Traceability
checking functional, 284
linking principles together using rationales and implications, 126
qualities of good concerns, 116
Tradeoff points, in ATAM, 229
Tradeoffs, consider evolution, 552
Training, development resource concerns, 575
Transaction logs, 528–529
Transaction stores, 295
Transactional consistency
backup and restore planning, 400–401
relax for availability and resilience, 532
relax to improve performance/scalability, 501–502, 507
Transactions
avoiding overhead, 507
as sequence of data updates, 306
Transient scalability, 480
Transition entities, in state model, 347
Trust and permissions model. *See also* Security policies, 319
Turnaround time class, response time, 478–479
Twin Peaks model (Nuseibeh), 87
Two-tier client server approach, 16–17

U

UML (unified modeling language)
activity diagram of details in architecture definition, 93
activity diagram of supporting activities in architecture definition, 89–90
ATAM process diagram, 227
codeline models in, 366–367

UML, *continued*
common design models in, 363
component diagrams for Functional views, 273–275
context diagram, 257
deployment diagram for network model, 384
deployment diagram for runtime platform model, 379–381
estimating platform availability, 519
information flow models, 317
as modeling language, 185–186
module structure models in, 360–361
paper-based scenario models, 154–155
presentation of AD, 214
state diagram for information lifecycle model, 318
statecharts in, 317–318, 348–350
static and dynamic elements represented in, 183
static information structure models in, 311–313
system-level concurrency models in, 342–343
use cases in, 146, 198
Unplanned downtime, 514–515
Updating
keeping AD current, 204
models, 193
unavoidable multiple updaters, 324
Upgrades
development resource concerns, 575–576
installation model, 402–403
operational concerns, 394
Uptime, global availability requirements and, 539
Usability perspective
activities, 598–599
applicability to views, 596–597
architectural tactics, 599
concerns, 597–598
defined, 568
desired quality, 438
overview of, 595–596
problems and pitfalls, 599–600
Use cases
for documenting functional scenarios, 146
UML context diagram, 257–258
walkthroughs for context model, 260
Users
authentication of, 49
classes of stakeholders, 134, 137
expanding focus to include all stakeholders not just end users, 2
visibility of identifiers to, 301

V

Validation
of abstraction, 218
of assumptions, 218–219
of models, 193
perspectives in, 57–58
scenarios in, 147
Value sets, identify configuration, 409–414
Variation points, 556–558
Viewpoint catalog
view relationships in, 243–244
viewpoint definitions in, 244–245
Viewpoints. *See also* by individual types
in AD, 210–211
benefits of, 38
catalog of core, 39–42
comparing Functional, Information, and Operational viewpoints, 46–47
comparing views/viewpoints with objects/classes, 35
comparing with views and perspectives, 59
deciding what to include in a view, 34–35
function of, 6
ISO recommendations for documenting an architecture, 206–207
overview of, 36–37
perspectives applied to, 51–54
perspectives as means of modifying/enhancing, 47
perspectives compared with, 58–60
pitfalls of, 39
relationship between core concepts, 37–38
in software architecture example, 4–5
summary and further reading, 43
user authentication and, 49
view relationships, in viewpoint catalog, 243–244
views based on, 178

Views
in AD, 210–211
AD partitioned into, 33
based on viewpoints, 178
benefits of, 38
comparing views/viewpoints with objects/classes, 35
comparing with perspectives, 59
comparing with viewpoints and perspectives, 59
consisting of one or more models, 178
deciding what to include in, 34–35
defined, 34
designing system architecture and, 31–34
example comparing Functional, Information, and Operational viewpoints, 46–47
function of, 6
important views for typical systems, 42
ISO recommendations for documenting an architecture, 206–207
overview of, 34–35
perspectives applied to, 51–54
perspectives as means of modifying/enhancing, 47
perspectives compared with, 45
pitfall of wrong set of, 39
relationship between core concepts, 37–38
in software architecture example, 4–5
summary and further reading, 43
view relationships, in viewpoint catalog, 243–244
view-specific principles, 210
Views, applying perspectives to
Accessibility perspective, 569–570
Availability and Resilience perspective, 512–513
Development Resource perspective, 574–575
Evolution perspective, 544–545
Internationalization perspective, 580–581
Location perspective, 585–586
Performance and Scalability perspective, 476–477
Regulation perspective, 591–592
Security perspective, 441–442
Usability perspective, 596–597
Views, consistency across
Concurrency and Deployment views, 433–434
Concurrency and Development views, 433
Context and Deployment views, 428
Context and Functional views, 427
Context and Information views, 427–428
Deployment and Operational views, 434
Functional and Concurrency views, 429, 431
Functional and Deployment views, 430–431
Functional and Development views, 430
Functional and Information views, 428–429
Functional and Operational views, 431
Information and Deployment views, 432
Information and Operational views, 432–433
operational monitoring and control concerns, 425–426
relationships between views, 426–427
Viewtypes, SEI (Software Engineering Institute) "Views and Beyond" Views, 624
Visible behaviors
in airline reservation example, 15–18
internal organization and, 19
system properties, 13–14
Voice recognition system, for visually impaired users, 571
Volumetrics, 329

W

W3C (World Wide Web Consortium), 113
Walkthroughs
activities of Functional viewpoint, 284
for architectural definition, 232
context model and, 260
for defining scope and exploring options, 231
for evaluating architecture, 220–222
for functional scenarios, 284
for scenarios, 155–156
for system construction, 232
Waterfall approach
plan-driven methods, 610
to software development, 99–100

Weakest link, securing, 457, 465–466
Wiki documentation of AD, 213–214
Workflows, in addressing information quality, 308
Workload
- analyzing performance models, 488
- avoiding unavailability through overload, 535
- consolidating, 494–495, 507
- consolidating related, 494–495
- degrading gracefully, 499–500
- distributing processing over time, 495–496
- optimizing repeated processing, 491–492
- partitioning by relaxing transactional consistency, 501
- peak load behavior and, 481–482
- predicting system performance, 476
- prioritizing processing, 493–494
- reducing risk of transaction overhead, 507
- responsiveness to, 478
- scale up or scale out, 498–499
- throughput and, 479–480

World Wide Web Consortium (W3C), 113

X

XP (Extreme Programming), 101, 547, 607

Y

Yourdon model, 276