FOUNDATIONS OF SOFTWARE AND SYSTEM PERFORMANCE ENGINEERING

Process, Performance Modeling, Requirements, Testing, Scalability, and Practice

André B. Bondi

FREE SAMPLE CHAPTER

SHARE WITH OTHERS
Praise for *Foundations of Software and System Performance Engineering*

“If this book had only been available to the contractors building healthcare.gov, and they read and followed the lifecycle performance processes, there would not have been the enormous problems apparent in that health care application. In my 40-plus years of experience in building leading-edge software and hardware products, poor performance is the single most frequent cause of the failure or cancellation of a new, software-intensive project. Performance requirements are often neglected or poorly formulated during the planning and requirements phases of a project. Consequently, the software architecture and the resulting delivered system are unable to meet performance needs. This book provides the reader with the techniques and skills necessary to implement performance engineering at the beginning of a project and manage those requirements throughout the lifecycle of the product. I cannot recommend this book highly enough.”

— Don Shafer, CSDP, Technical Fellow, Athens Group, LLC

“Well written and packed with useful examples, *Foundations of Software and System Performance Engineering* provides a thorough presentation of this crucial topic. Drawing upon decades of professional experience, Dr. Bondi shows how the principles of performance engineering can be applied in many different fields and disciplines.”

— Matthew Scarpino, author of Programming the Cell Processor and OpenCL in Action
Foundations of Software and System Performance Engineering
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 author 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.

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

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

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data
Bondi, André B., author.
Foundations of software and system performance engineering : process, performance modeling, requirements, testing, scalability, and practice / André B. Bondi.
pages cm
Includes bibliographical references and index.
QA76.9.E94B66 2015
005.1’4—dc23
2014020070

Copyright © 2015 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.


Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, August 2014
In memory of my father, Henry S. Bondi, who liked eclectic solutions to problems, and of my violin teacher, Fritz Rikko, who taught me how to analyze and debug.

À tous qui ont attendu.
This page intentionally left blank
Contents

Preface xxiii
Acknowledgments xxix
About the Author xxxi

Chapter 1 Why Performance Engineering?
   Why Performance Engineers? 1
      1.1 Overview 1
      1.2 The Role of Performance Requirements in Performance Engineering 4
      1.3 Examples of Issues Addressed by Performance Engineering Methods 5
      1.4 Business and Process Aspects of Performance Engineering 6
      1.5 Disciplines and Techniques Used in Performance Engineering 8
      1.6 Performance Modeling, Measurement, and Testing 10
      1.7 Roles and Activities of a Performance Engineer 11
      1.8 Interactions and Dependencies between Performance Engineering and Other Activities 13
      1.9 A Road Map through the Book 15
      1.10 Summary 17

Chapter 2 Performance Metrics 19
      2.1 General 19
      2.2 Examples of Performance Metrics 23
      2.3 Useful Properties of Performance Metrics 24
      2.4 Performance Metrics in Different Domains 26
### Chapter 2: Performance Metrics

#### 2.4.1 Conveyor in a Warehouse
27

#### 2.4.2 Fire Alarm Control Panel
28

#### 2.4.3 Train Signaling and Departure Boards
29

#### 2.4.4 Telephony
30

#### 2.4.5 An Information Processing Example: Order Entry and Customer Relationship Management
30

#### 2.5 Examples of Explicit and Implicit Metrics
32

#### 2.6 Time Scale Granularity of Metrics
32

#### 2.7 Performance Metrics for Systems with Transient, Bounded Loads
33

#### 2.8 Summary
35

#### 2.9 Exercises
35

### Chapter 3: Basic Performance Analysis

#### 3.1 How Performance Models Inform Us about Systems
37

#### 3.2 Queues in Computer Systems and in Daily Life
38

#### 3.3 Causes of Queueing
39

#### 3.4 Characterizing the Performance of a Queue
42

#### 3.5 Basic Performance Laws: Utilization Law, Little’s Law
45

##### 3.5.1 Utilization Law
45

##### 3.5.2 Little’s Law
47

#### 3.6 A Single-Server Queue
49

#### 3.7 Networks of Queues: Introduction and Elementary Performance Properties
52

##### 3.7.1 System Features Described by Simple Queueing Networks
53

##### 3.7.2 Quantifying Device Loadings and Flow through a Computer System
54

##### 3.7.3 Upper Bounds on System Throughput
56

##### 3.7.4 Lower Bounds on System Response Times
58

#### 3.8 Open and Closed Queueing Network Models
58

##### 3.8.1 Simple Single-Class Open Queueing Network Models
59
3.8.2 Simple Single-Class Closed Queueing Network Model 60
3.8.3 Performance Measures and Queueing Network Representation: A Qualitative View 62
3.9 Bottleneck Analysis for Single-Class Closed Queueing Networks 63
  3.9.1 Asymptotic Bounds on Throughput and Response Time 63
  3.9.2 The Impact of Asynchronous Activity on Performance Bounds 66
3.10 Regularity Conditions for Computationally Tractable Queueing Network Models 68
3.11 Mean Value Analysis of Single-Class Closed Queueing Network Models 69
3.12 Multiple-Class Queueing Networks 71
3.13 Finite Pool Sizes, Lost Calls, and Other Lost Work 75
3.14 Using Models for Performance Prediction 77
3.15 Limitations and Applicability of Simple Queueing Network Models 78
3.16 Linkage between Performance Models, Performance Requirements, and Performance Test Results 79
3.17 Applications of Basic Performance Laws to Capacity Planning and Performance Testing 80
3.18 Summary 80
3.19 Exercises 81

Chapter 4 Workload Identification and Characterization 85
  4.1 Workload Identification 85
  4.2 Reference Workloads for a System in Different Environments 87
  4.3 Time-Varying Behavior 89
  4.4 Mapping Application Domains to Computer System Workloads 91
    4.4.1 Example: An Online Securities Trading System for Account Holders 91
Chapter 4  From Workloads to Performance Requirements

4.4.2 Example: An Airport Conveyor System  92
4.4.3 Example: A Fire Alarm System  94
4.5 Numerical Specification of the Workloads  95
4.5.1 Example: An Online Securities Trading System for Account Holders  96
4.5.2 Example: An Airport Conveyor System  97
4.5.3 Example: A Fire Alarm System  98
4.6 Numerical Illustrations  99
4.6.1 Numerical Data for an Online Securities Trading System  100
4.6.2 Numerical Data for an Airport Conveyor System  101
4.6.3 Numerical Data for the Fire Alarm System  102
4.7 Summary  103
4.8 Exercises  103

Chapter 5  From Workloads to Business Aspects of Performance Requirements  105

5.1 Overview  105
5.2 Performance Requirements and Product Management  106
5.2.1 Sizing for Different Market Segments: Linking Workloads to Performance Requirements  107
5.2.2 Performance Requirements to Meet Market, Engineering, and Regulatory Needs  108
5.2.3 Performance Requirements to Support Revenue Streams  110
5.3 Performance Requirements and the Software Lifecycle  111
5.4 Performance Requirements and the Mitigation of Business Risk  112
5.5 Commercial Considerations and Performance Requirements  114
5.5.1 Performance Requirements, Customer Expectations, and Contracts  114
5.5.2 System Performance and the Relationship between Buyer and Supplier  114
Contents

5.5.3 Confidentiality 115
5.5.4 Performance Requirements and the Outsourcing of Software Development 116
5.5.5 Performance Requirements and the Outsourcing of Computing Services 116

5.6 Guidelines for Specifying Performance Requirements 116
5.6.1 Performance Requirements and Functional Requirements 117
5.6.2 Unambiguousness 117
5.6.3 Measurability 118
5.6.4 Verifiability 119
5.6.5 Completeness 119
5.6.6 Correctness 120
5.6.7 Mathematical Consistency 120
5.6.8 Testability 120
5.6.9 Traceability 121
5.6.10 Granularity and Time Scale 122

5.7 Summary 122
5.8 Exercises 123

Chapter 6 Qualitative and Quantitative Types of Performance Requirements 125

6.1 Qualitative Attributes Related to System Performance 126
6.2 The Concept of Sustainable Load 127
6.3 Formulation of Response Time Requirements 128
6.4 Formulation of Throughput Requirements 130
6.5 Derived and Implicit Performance Requirements 131
   6.5.1 Derived Performance Requirements 132
   6.5.2 Implicit Requirements 132
6.6 Performance Requirements Related to Transaction Failure Rates, Lost Calls, and Lost Packets 134
6.7 Performance Requirements Concerning Peak and Transient Loads 135
Chapter 7 Eliciting, Writing, and Managing Performance Requirements

7.1 Elicitation and Gathering of Performance Requirements 140
7.2 Ensuring That Performance Requirements Are Enforceable 143
7.3 Common Patterns and Antipatterns for Performance Requirements 144
  7.3.1 Response Time Pattern and Antipattern 144
  7.3.2 “… All the Time/… of the Time” Antipattern 145
  7.3.3 Resource Utilization Antipattern 146
  7.3.4 Number of Users to Be Supported Pattern/Antipattern 146
  7.3.5 Pool Size Requirement Pattern 147
  7.3.6 Scalability Antipattern 147
7.4 The Need for Mathematically Consistent Requirements: Ensuring That Requirements Conform to Basic Performance Laws 148
7.5 Expressing Performance Requirements in Terms of Parameters with Unknown Values 149
7.6 Avoidance of Circular Dependencies 149
7.7 External Performance Requirements and Their Implications for the Performance Requirements of Subsystems 150
7.8 Structuring Performance Requirements Documents 150
7.9 Layout of a Performance Requirement 153
7.10 Managing Performance Requirements: Responsibilities of the Performance Requirements Owner 155
7.11 Performance Requirements Pitfall: Transition from a Legacy System to a New System 156
7.12 Formulating Performance Requirements to Facilitate Performance Testing 158
7.13 Storage and Reporting of Performance Requirements 160
7.14 Summary 161

Chapter 8 System Measurement Techniques and Instrumentation 163

8.1 General 163
8.2 Distinguishing between Measurement and Testing 167
8.3 Validate, Validate, Validate; Scrutinize, Scrutinize, Scrutinize 168
8.4 Resource Usage Measurements 168
  8.4.1 Measuring Processor Usage 169
  8.4.2 Processor Utilization by Individual Processes 171
  8.4.3 Disk Utilization 173
  8.4.4 Bandwidth Utilization 174
  8.4.5 Queue Lengths 175
8.5 Utilizations and the Averaging Time Window 175
8.6 Measurement of Multicore or Multiprocessor Systems 177
8.7 Measuring Memory-Related Activity 180
  8.7.1 Memory Occupancy 181
  8.7.2 Paging Activity 181
8.8 Measurement in Production versus Measurement for Performance Testing and Scalability 181
8.9 Measuring Systems with One Host and with Multiple Hosts 183
  8.9.1 Clock Synchronization of Multiple Hosts 184
  8.9.2 Gathering Measurements from Multiple Hosts 184
8.10 Measurements from within the Application 186
8.11 Measurements in Middleware 187
8.12 Measurements of Commercial Databases 188
8.13 Response Time Measurements 189
8.14 Code Profiling 190
8.15 Validation of Measurements Using Basic Properties of Performance Metrics 191
Chapter 9 Performance Testing

9.1 Overview of Performance Testing 199
9.2 Special Challenges 202
9.3 Performance Test Planning and Performance Models 203
9.4 A Wrong Way to Evaluate Achievable System Throughput 208
9.5 Provocative Performance Testing 209
9.6 Preparing a Performance Test 210
  9.6.1 Understanding the System 211
  9.6.2 Pilot Testing, Playtime, and Performance Test Automation 213
  9.6.3 Test Equipment and Test Software Must Be Tested, Too 213
  9.6.4 Deployment of Load Drivers 214
  9.6.5 Problems with Testing Financial Systems 216
9.7 Lab Discipline in Performance Testing 217
9.8 Performance Testing Challenges Posed by Systems with Multiple Hosts 218
9.9 Performance Testing Scripts and Checklists 219
9.10 Best Practices for Documenting Test Plans and Test Results 220
9.11 Linking the Performance Test Plan to Performance Requirements 222
9.12 The Role of Performance Tests in Detecting and Debugging Concurrency Issues 223
9.13 Planning Tests for System Stability 225
9.14 Prospective Testing When Requirements Are Unspecified 226
9.15 Structuring the Test Environment to Reflect the Scalability of the Architecture 228
9.16 Data Collection 229
9.17 Data Reduction and Presentation 230
9.18 Interpreting the Test Results 231
9.18.1 Preliminaries 231
9.18.2 Example: Services Use Cases 231
9.18.3 Example: Transaction System with High Failure Rate 235
9.18.4 Example: A System with Computationally Intense Transactions 237
9.18.5 Example: System Exhibiting Memory Leak and Deadlocks 241
9.19 Automating Performance Tests and the Analysis of the Outputs 244
9.20 Summary 246
9.21 Exercises 246

Chapter 10 System Understanding, Model Choice, and Validation 251
10.1 Overview 252
10.2 Phases of a Modeling Study 254
10.3 Example: A Conveyor System 256
10.4 Example: Modeling Asynchronous I/O 260
10.5 Systems with Load-Dependent or Time-Varying Behavior 266
10.5.1 Paged Virtual Memory Systems That Thrash 266
10.5.2 Applications with Increasing Processing Time per Unit of Work 267
10.5.3 Scheduled Movement of Load, Periodic Loads, and Critical Peaks 267
10.6 Summary 268
10.7 Exercises 270
Chapter 11  Scalability and Performance  273

11.1  What Is Scalability?  273

11.2  Scaling Methods  275
   11.2.1  Scaling Up and Scaling Out  276
   11.2.2  Vertical Scaling and Horizontal Scaling  276

11.3  Types of Scalability  277
   11.3.1  Load Scalability  277
   11.3.2  Space Scalability  279
   11.3.3  Space-Time Scalability  280
   11.3.4  Structural Scalability  281
   11.3.5  Scalability over Long Distances and under Network Congestion  281

11.4  Interactions between Types of Scalability  282

11.5  Qualitative Analysis of Load Scalability and Examples  283
   11.5.1  Serial Execution of Disjoint Transactions and the Inability to Exploit Parallel Resources  283
   11.5.2  Busy Waiting on Locks  286
   11.5.3  Coarse Granularity Locking  287
   11.5.4  Ethernet and Token Ring: A Comparison  287
   11.5.5  Museum Checkrooms  289

11.6  Scalability Limitations in a Development Environment  292

11.7  Improving Load Scalability  293

11.8  Some Mathematical Analyses  295
   11.8.1  Comparison of Semaphores and Locks for Implementing Mutual Exclusion  296
   11.8.2  Museum Checkroom  298

11.9  Avoiding Scalability Pitfalls  299

11.10  Performance Testing and Scalability  302

11.11  Summary  303

11.12  Exercises  304
Chapter 12 Performance Engineering Pitfalls

12.1 Overview 308
12.2 Pitfalls in Priority Scheduling 308
12.3 Transient CPU Saturation Is Not Always a Bad Thing 312
12.4 Diminishing Returns with Multiprocessors or Multiple Cores 314
12.5 Garbage Collection Can Degrade Performance 315
12.6 Virtual Machines: Panacea or Complication? 315
12.7 Measurement Pitfall: Delayed Time Stamping and Monitoring in Real-Time Systems 317
12.8 Pitfalls in Performance Measurement 318
12.9 Eliminating a Bottleneck Could Unmask a New One 319
12.10 Pitfalls in Performance Requirements Engineering 321
12.11 Organizational Pitfalls in Performance Engineering 321
12.12 Summary 322
12.13 Exercises 323

Chapter 13 Agile Processes and Performance Engineering

13.1 Overview 325
13.2 Performance Engineering under an Agile Development Process 327
  13.2.1 Performance Requirements Engineering Considerations in an Agile Environment 328
  13.2.2 Preparation and Alignment of Performance Testing with Sprints 329
  13.2.3 Agile Interpretation and Application of Performance Test Results 330
  13.2.4 Communicating Performance Test Results in an Agile Environment 331
13.3 Agile Methods in the Implementation and Execution of Performance Tests 332
  13.3.1 Identification and Planning of Performance Tests and Instrumentation 332
13.3.2 Using Scrum When Implementing Performance Tests and Purpose-Built Instrumentation 333
13.3.3 Peculiar or Irregular Performance Test Results and Incorrect Functionality May Go Together 334
13.4 The Value of Playtime in an Agile Performance Testing Process 334
13.5 Summary 336
13.6 Exercises 336

Chapter 14 Working with Stakeholders to Learn, Influence, and Tell the Performance Engineering Story 339
14.1 Determining What Aspect of Performance Matters to Whom 340
14.2 Where Does the Performance Story Begin? 341
14.3 Identification of Performance Concerns, Drivers, and Stakeholders 344
14.4 Influencing the Performance Story 345
14.4.1 Using Performance Engineering Concerns to Affect the Architecture and Choice of Technology 345
14.4.2 Understanding the Impact of Existing Architectures and Prior Decisions on System Performance 346
14.4.3 Explaining Performance Concerns and Sharing and Developing the Performance Story with Different Stakeholders 347
14.5 Reporting on Performance Status to Different Stakeholders 353
14.6 Examples 354
14.7 The Role of a Capacity Management Engineer 355
14.8 Example: Explaining the Role of Measurement Intervals When Interpreting Measurements 356
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.9 Ensuring Ownership of Performance Concerns and Explanations by Diverse Stakeholders</td>
<td>360</td>
</tr>
<tr>
<td>14.10 Negotiating Choices for Design Changes and Recommendations for System Improvement among Stakeholders</td>
<td>360</td>
</tr>
<tr>
<td>14.11 Summary</td>
<td>362</td>
</tr>
<tr>
<td>14.12 Exercises</td>
<td>363</td>
</tr>
</tbody>
</table>

**Chapter 15 Where to Learn More**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1 Overview</td>
<td>367</td>
</tr>
<tr>
<td>15.2 Conferences and Journals</td>
<td>369</td>
</tr>
<tr>
<td>15.3 Texts on Performance Analysis</td>
<td>370</td>
</tr>
<tr>
<td>15.4 Queueing Theory</td>
<td>372</td>
</tr>
<tr>
<td>15.5 Discrete Event Simulation</td>
<td>372</td>
</tr>
<tr>
<td>15.6 Performance Evaluation of Specific Types of Systems</td>
<td>373</td>
</tr>
<tr>
<td>15.7 Statistical Methods</td>
<td>374</td>
</tr>
<tr>
<td>15.8 Performance Tuning</td>
<td>374</td>
</tr>
<tr>
<td>15.9 Summary</td>
<td>375</td>
</tr>
</tbody>
</table>

** References**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>385</td>
</tr>
</tbody>
</table>
Preface

The performance engineering of computer systems and the systems they control concerns the methods, practices, and disciplines that may be used to ensure that the systems provide the performance that is expected of them. Performance engineering is a process that touches every aspect of the software lifecycle, from conception and requirements planning to testing and delivery. Failure to address performance concerns at the beginning of the software lifecycle significantly increases the risk that a software project will fail. Indeed, performance is the single largest risk to the success of any software project. Readers in the United States will recall that poor performance was the first sign that healthcare.gov, the federal web site for obtaining health insurance policies that went online in late 2013, was having a very poor start. News reports indicate that the processes and steps recommended in this book were not followed during its development and rollout. Performance requirements were inadequately specified, and there was almost no performance testing prior to the rollout because time was not available for it. This should be a warning that adequate planning and timely scheduling are preconditions for the successful incorporation of performance engineering into the software development lifecycle. “Building and then tuning” is an almost certain recipe for performance failure.

Scope and Purpose

The performance of a system is often characterized by the amount of time it takes to accomplish a variety of prescribed tasks and the number of times it can accomplish those tasks in a set time period. For example:

- A government system for selling health insurance policies to the general public, such as healthcare.gov, would be expected to determine an applicant’s eligibility for coverage, display available options, and confirm the choice of policy and the
premium due within designated amounts of time regardless of how many applications were to be processed within the peak hour.

- An online stock trading system might be expected to obtain a quote of the current value of a security within a second or so and execute a trade within an even shorter amount of time.
- A monitoring system, such as an alarm system, is expected to be able to process messages from a set of sensors and display corresponding status indications on a console within a short time of their arrival.
- A web-based news service would be expected to retrieve a story and display related photographs quickly.

This is a book about the practice of the performance engineering of software systems and software-controlled systems. It will help the reader address the following performance-related questions concerning the architecture, development, testing, and sizing of a computer system or a computer-controlled system:

- What capacity should the system have? How do you specify that capacity in both business-related and engineering terms?
- What business, social, and engineering needs will be satisfied by given levels of throughput and system response time?
- How many data records, abstract objects, or representations of concrete, tangible objects must the system be able to manage, monitor, and store?
- What metrics do you use to describe the performance your system needs and the performance it has?
- How do you specify the performance requirements of a system? Why do you need to specify them in the first place?
- How can the resource usage performance of a system be measured? How can you verify the accuracy of the measurements?
- How can you use mathematical models to predict a system’s performance? Can the models be used to predict the performance if an application is added to the system or if the transaction rate increases?
- How can mathematical models of performance be used to plan performance tests and interpret the results?
• How can you test performance in a manner that tells you if the system is functioning properly at all load levels and if it will scale to the extent and in the dimensions necessary?

• What can poor performance tell you about how the system is functioning?

• How do you architect a system to be scalable? How do you specify the dimensions and extent of the scalability that will be required now or in the future? What architecture and design features undermine the scalability of a system?

• Are there common performance mistakes and misconceptions? How do you avoid them?

• How do you incorporate performance engineering into an agile development process?

• How do you tell the performance story to management?

Questions like these must be addressed at every phase of the software lifecycle. A system is unlikely to provide adequate performance with a cost-effective configuration unless its architecture is influenced by well-formulated, testable performance requirements. The requirements must be written in measurable, unambiguous, testable terms. Performance models may be used to predict the effects of design choices such as the use of scheduling rules and the deployment of functions on one or more hosts. Performance testing must be done to ensure that all system components are able to meet their respective performance needs, and to ensure that the end-to-end performance of the system meets user expectations, the owner’s expectations, and, where applicable, industry and government regulations. Performance requirements must be written to help the architects identify the architectural and technological choices needed to ensure that performance needs are met. Performance requirements should also be used to determine how the performance of a system will be tested.

The need for performance engineering and general remarks about how it is practiced are presented in Chapter 1. Metrics are needed to describe performance quantitatively. A discussion of performance metrics is given in Chapter 2. Once performance metrics have been identified, basic analysis methods may be used to make predictions about system performance, as discussed in Chapter 3. The anticipated workload can be quantitatively described as in Chapter 4, and performance requirements can be specified. Necessary attributes of performance
requirements and best practices for writing and managing them are discussed in Chapters 5 through 7. To understand the performance that has been attained and to verify that performance requirements have been met, the system must be measured. Techniques for doing so are given in Chapter 8. Performance tests should be structured to enable the evaluation of the scalability of a system, to determine its capacity and responsiveness, and to determine whether it is meeting throughput and response time requirements. It is essential to test the performance of all components of the system before they are integrated into a whole, and then to test system performance from end to end before the system is released. Methods for planning and executing performance tests are discussed in Chapter 9. In Chapter 10 we discuss procedures for evaluating the performance of a system and the practice of performance modeling with some examples. In Chapter 11 we discuss ways of describing system scalability and examine ways in which scalability is enhanced or undermined. Performance engineering pitfalls are examined in Chapter 12, and performance engineering in an agile context is discussed in Chapter 13. In Chapter 14 we consider ways of communicating the performance story. Chapter 15 contains a discussion of where to learn more about various aspects of performance engineering.

This book does not contain a presentation of the elements of probability and statistics and how they are applied to performance engineering. Nor does it go into detail about the mathematics underlying some of the main tools of performance engineering, such as queueing theory and queueing network models. There are several texts that do this very well already. Some examples of these are mentioned in Chapter 15, along with references on some detailed aspects of performance engineering, such as database design. Instead, this book focuses on various steps of the performance engineering process and the link between these steps and those of a typical software lifecycle. For example, the chapters on performance requirements engineering draw parallels with the engineering of functional requirements, and the chapter on scalability explains how performance models can be used to evaluate it and how architectural characteristics might affect it.
Audience

This book will be of interest to software and system architects, requirements engineers, designers and developers, performance testers, and product managers, as well as their managers. While all stakeholders should benefit from reading this book from cover to cover, the following stakeholders may wish to focus on different subsets of the book to begin with:

- Product owners and product managers who are reluctant to make commitments to numerical descriptions of workloads and requirements will benefit from the chapters on performance metrics, workload characterization, and performance requirements engineering.
- Functional testers who are new to performance testing may wish to read the chapters on performance metrics, performance measurement, performance testing, basic modeling, and performance requirements when planning the implementation of performance tests and testing tools.
- Architects and developers who are new to performance engineering could begin by reading the chapters on metrics, basic performance modeling, performance requirements engineering, and scalability.

This book may be used as a text in a senior- or graduate-level course on software performance engineering. It will give the students the opportunity to learn that computer performance evaluation involves integrating quantitative disciplines with many aspects of software engineering and the software lifecycle. These include understanding and being able to explain why performance is important to the system being built, the commercial and engineering implications of system performance, the architectural and software aspects of performance, the impact of performance requirements on the success of the system, and how the performance of the system will be tested.
This page intentionally left blank
Acknowledgments

This book is based in part on a training course entitled Foundations of Performance Engineering. I developed this course to train performance engineering and performance testing teams at various Siemens operating companies. The course may be taught on its own or, as my colleague Alberto Avritzer and I have done, as part of a consulting engagement. When teaching the course as part of a consulting engagement, one may have the opportunity to integrate the client’s performance issues and even test data into the class material. This helps the clients resolve the particular issues they face and is effective at showing how the material on performance engineering presented here can be integrated into their software development processes.

One of my goals in writing this book was to relate this practical experience to basic performance modeling methods and to link performance engineering methods to the various stages of the software lifecycle. I was encouraged to write it by Dr. Dan Paulish, my first manager at Siemens Corporate Research (now Siemens Corporation, Corporate Technology, or SC CT); by Prof. Len Bass, who at the time was with the Software Engineering Institute in Pittsburgh; and by Prof. C. Murray Woodside of Carleton University in Ottawa. We felt that there was a teachable story to tell about the practical performance issues I have encountered during a career in performance engineering that began during the heyday of mainframe computers.

My thinking on performance requirements has been strongly influenced by Brian Berenbach, who has been a driving force in the practice of requirements engineering at SC CT. I would like to thank my former AT&T Labs colleagues, Dr. David Hoeflin and Dr. Richard Oppenheim, for reading and commenting on selected chapters. We worked together for many years as part of a large group of performance specialists. My experience in that group was inspiring and rewarding. I would also like to thank Dr. Alberto Avritzer of SC CT for many lively discussions on performance engineering.

I would like to thank the following past and present managers and staff at SC CT for their encouragement in the writing of this book.
Between them, Raj Varadarajan and Dr. Michael Golm read all of the chapters of the book and made useful comments before submission to the publisher.

Various Siemens operating units with whom I have worked on performance issues kindly allowed me to use material I had prepared for them in published work. Ruth Weitzenfeld, SC CT’s librarian, cheerfully obtained copies of many references. Patti Schmidt, SC CT’s in-house counsel, arranged for permission to quote from published work I had prepared while working at Siemens. Dr. Yoni Levi of AT&T Labs kindly arranged for me to obtain AT&T’s permission to quote from a paper I had written on scalability while working there. This paper forms the basis for much of the content of Chapter 11.

I would like to thank my editors at Addison-Wesley, Bernard Goodwin and Chris Zahn, and their assistant, Michelle Housley, for their support in the preparation of this book. It has been a pleasure to work with them. The copy editor, Barbara Wood, highlighted several points that needed clarification. Finally, the perceptive comments of the publisher’s reviewers, Nick Rozanski, Don Shafer, and Matthew Scarpino, have done much to make this a better book.
About the Author

André B. Bondi is a Senior Staff Engineer working in performance engineering at Siemens Corp., Corporate Technology, in Princeton, New Jersey. He has worked on performance issues in several domains, including telecommunications, conveyor systems, finance systems, building surveillance, railways, and network management systems. Prior to joining Siemens, he held senior performance positions at two start-up companies. Before that, he spent more than ten years working on a variety of performance and operational issues at AT&T Labs and its predecessor, Bell Labs. He has taught courses in performance, simulation, operating systems principles, and computer architecture at the University of California, Santa Barbara. Dr. Bondi holds a PhD in computer science from Purdue University, an MSc in statistics from University College London, and a BSc in mathematics from the University of Exeter.
This page intentionally left blank
Chapter 1

Why Performance Engineering? Why Performance Engineers?

This chapter describes the importance of performance engineering in a software project and explains the role of a performance engineer in ensuring that the system has good performance upon delivery. Overviews of different aspects of performance engineering are given.

1.1 Overview

The performance of a computer-based system is often characterized by its ability to perform defined sets of activities at fast rates and with quick response time. Quick response times, speed, and scalability are highly desired attributes of any computer-based system. They are also competitive differentiators. That is, they are attributes that distinguish a system from other systems with like functionality and make it more attractive to a prospective buyer or user.
If a system component has poor performance, the system as a whole may not be able to function as intended. If a system has poor performance, it will be unattractive to prospective users and buyers. If the project fails as a result, the investment in building the system will have been wasted. The foregoing is true whether the system is a command and control system, a transaction-based system, an information retrieval system, a video game, an entertainment system, a system for displaying news, or a system for streaming media.

The importance of performance may be seen in the following examples:

- A government-run platform for providing services on a grand scale must be able to handle a large volume of transactions from the date it is brought online. If it is not able to do so, it will be regarded as ineffective. In the United States, the federal web site for applying for health insurance mandated by the Affordable Care Act, healthcare.gov, was extremely slow for some time after it was made available to the public. According to press reports and testimony before the United States Congress, functional, capacity, and performance requirements were unclear. Moreover, the system was not subjected to rigorous performance tests before being brought online [Eilperin2013].

- An online securities trading system must be able to handle large numbers of transactions per second, especially in a volatile market with high trading volume. A brokerage house whose system cannot do this will lose business very quickly, because slow execution could lead to missing a valuable trading opportunity.

- An online banking system must display balances and statements rapidly. It must acknowledge transfers and the transmission of payments quickly for users to be confident that these transactions have taken place as desired.

- Regulations such as fire codes require that audible and visible alarms be triggered within 10 seconds of smoke being detected. In many jurisdictions, a building may not be used if the fire alarm system cannot meet this requirement.

- A telephone network must be able to handle large numbers of call setups and tear downs per second and provide such services as call forwarding and fraud detection within a short time of each call being initiated.
• A rail network control system must be able to monitor train movements and set signals and switches accordingly within very short amounts of time so that trains are routed to their correct destinations without colliding with one another.

• In combat, a system that has poor performance may endanger the lives or property of its users instead of endangering those of the enemy.

• A medical records system must be able to pull up patient records and images quickly so that retrieval will not take too much of a doctor’s time away from diagnosis and treatment.

The foregoing examples show that performance is crucial to the correct functioning of a software system and of the application it controls. As such, performance is an attribute of system quality that presents significant business and engineering risks. In some applications, such as train control and fire alarm systems, it is also an essential ingredient of safety. Performance engineering mitigates these risks by ensuring that adequate attention is paid to them at every stage of the software lifecycle, while improving the capacity of systems, improving their response times, ensuring their scalability, and increasing user productivity. All of these are key competitive differentiators for any software product.

Despite the importance of system performance and the severe risk associated with inattentiveness to it, it is often ignored until very late in the software development cycle. Too often, the view is that performance objectives can be achieved by tuning the system once it is built. This mindset of “Build it, then tune it” is a recurring cause of the failure of a system to meet performance needs [SmithWilliams2001]. Most performance problems have their root causes in poor architectural choices. For example:

• An architectural choice could result in the creation of foci of overload.

• A decision is made that a set of operations that could be done in parallel on a multiprocessor or multicore host will be handled by a single thread. This would result in the onset of a software bottleneck for sufficiently large loads.

One of the possible consequences of detecting a performance issue with an architectural cause late in the software lifecycle is that a considerable amount of implementation work must be undone and redone.
This is needlessly expensive when one considers that the problem could have been avoided by performing an architectural review. This also holds for other quality attributes such as reliability, availability, and security.

1.2 The Role of Performance Requirements in Performance Engineering

To ensure that performance needs are met, it is important that they be clearly specified in requirements early in the software development cycle. Early and concise specifications of performance requirements are necessary because:

- Performance requirements are potential drivers of the system architecture and the choice of technologies to be used in the system’s implementation. Moreover, many performance failures have their roots in poor architectural choices. Modification of the architecture before a system is implemented is cheaper than rebuilding a slow system from scratch.

- Performance requirements are closely related to the contractual expectations of system performance negotiated between buyer and seller, as well as to any relevant regulatory requirements such as those for fire alarm systems.

- The performance requirements will be reflected in the performance test plan.

- Drafting and reviewing performance requirements force the consideration of trade-offs between execution speed and system cost, as well as between execution speed and simplicity of both the architecture and the implementation. For instance, it is more difficult to design and correctly code a system that uses multithreading to achieve parallelism in execution than to build a single-threaded implementation.

- Development and/or hardware costs can be reduced if performance requirements that are found to be too stringent are relaxed early in the software lifecycle. For example, while a 1-second average response time requirement may be desirable, a 2-second requirement may be sufficient for business or engineering needs. Poorly specified performance
requirements can lead to confusion among stakeholders and the delivery of a poor-quality product with slow response times and inadequate capacity.

- If a performance issue that cannot be mapped to explicit performance requirements emerges during testing or production, stakeholders might not feel obliged to correct it.

We shall explore the principles of performance requirements in Chapter 5.

1.3 Examples of Issues Addressed by Performance Engineering Methods

Apart from mitigating business risk, performance engineering methods assist in answering a variety of questions about a software system. The performance engineer must frequently address questions related to capacity. For example:

- Can the system carry the peak load? The answer to this question depends on whether the system is adequately sized, and on whether its components can interact gracefully under load.

- Will the system cope with a surge in load and continue to function properly when the surge abates? This question is related to the reliability of the system. We do not want it to crash when it is most needed.

- What will be the performance impacts of adding new functionality to a system? To answer this question, we need to understand the extra work associated with each invocation of the functionality, and how often that functionality is invoked. We also need to consider whether the new functionality will adversely affect the performance of the system in its present form.

- Will the system be able to carry an increase in load? To answer this question, we must first ask whether there are enough resources to allow the system to perform at its current level.

- What is the performance impact of increasing the size of the user base? Answering this question entails understanding the memory and secondary storage footprints per user as well as in
total, and then being able to quantify the increased demand for memory, processing power, I/O, and network bandwidth.

- Can the system meet customer expectations or engineering needs if the average response time requirement is 2 seconds rather than 1 second? If so, it might be possible to build the system at a lower cost or with a simpler architecture. On the other hand, the choice of a simpler architecture could adversely affect the ability to scale up the offered load later, while still maintaining the response time requirement.

- Can the system provide the required performance with a cost-effective configuration? If it cannot, it will not fare well in the marketplace.

Performance can have an effect on the system’s functionality, or its perceived functionality. If the system does not respond to an action before there is a timeout, it may be declared unresponsive or down if timeouts occur in a sufficiently large number of consecutive attempts at the action.

The performance measures of healthy systems tend to behave in a predictable manner. Deviations from predictable performance are signs of potential problems. Trends or wild oscillations in the performance measurements may indicate that the system is unstable or that a crash will shortly occur. For example, steadily increasing memory occupancy indicates a leak that could bring the system down, while oscillating CPU utilization and average response times may indicate that the system is repeatedly entering deadlock and timing out.

### 1.4 Business and Process Aspects of Performance Engineering

Ensuring the performance of a system entails initial and ongoing investment. The investment is amply rewarded by reductions in business risk, increased system stability, and system scalability. Because performance is often the single biggest risk to the success of a project [Bass2007], reducing this risk will make a major contribution to reducing the total risk to the project overall.

The initial performance engineering investments in a software project include
Ensuring that there is performance engineering expertise on the project, perhaps including an individual designated as the lead performance engineer

Drafting performance requirements

Planning lab time for performance measurement and performance testing

Acquiring and preparing performance measurement tools, load generation tools, and analysis and reporting tools to simplify the presentation and tracking of the results of the performance tests

Incorporating sound performance engineering practices into every aspect of the software development cycle can considerably reduce the performance risk inherent in the development of a large, complicated system. The performance process should be harmonized with the requirements, architectural, development, and testing phases of the development lifecycle. In addition to the steps just described, the performance engineering effort should include

1. A review of the system architecture from the standpoints of performance, reliability, and scalability
2. An evaluation of performance characteristics of the technologies proposed in the architecture specification, including quick performance testing of any proposed platforms [MBH2005]
3. Incremental performance testing following incremental functional testing of the system, followed by suggestions for architectural and design revisions as needed
4. Retesting to overcome the issues revealed and remedied as a result of the previous step

Performance engineering methods can also be used to manage cost-effective system growth and added functionality. For an existing system, growth is managed by building a baseline model based on measurements of resource usage and query or other work unit rates taken at runtime. The baseline model is combined with projected traffic rates to determine resource requirements using mathematical models and other methods drawn from the field of operations research [LZGS1984, Kleinrock1975, Kleinrock1976, MenasceAlmeida2000].

We now turn to a discussion of the various disciplines and techniques a performance engineer can use to perform his or her craft.
1.5 Disciplines and Techniques Used in Performance Engineering

The practice of performance engineering draws on many disciplines and skills, ranging from the technological to the mathematical and even the political. Negotiating, listening, and writing skills are also essential for successful performance engineering, as is the case for successful architects and product owners. The set of original undergraduate major subjects taken by performance engineers the author has met includes such quantitative disciplines as mathematics, physics, chemical engineering, chemistry, biology, electrical engineering, statistics, economics, and operations research, as well as computer science. Those who have not majored in computer science will need to learn about such subjects as operating systems design, networking, and hardware architecture, while the computer scientists may need to acquire additional experience with working in a quantitative discipline.

To understand resource usage and information flow, the performance engineer must have at least a rudimentary knowledge of computer systems architecture, operating systems principles, concurrent programming principles, and software platforms such as web servers and database management systems. In addition, the performance engineer must have a sound grasp of the technologies and techniques used to measure resource usage and traffic demands, as well as those used to drive transactions through a system under test.

To understand performance requirements and the way the system will be used, it is necessary to know something about its domain of application. The performance and reliability needs of financial transaction systems, fire alarm systems, network management systems, conveyor belts, telecommunications systems, train control systems, online news services, search engines, and multimedia streaming services differ dramatically. For instance, the performance of fire alarm systems is governed by building and fire codes in the jurisdictions where the systems will be installed, while that of a telephone system may be governed by international standards. The performance needs of all the systems mentioned previously may be driven by commercial considerations such as competitive differentiation.

Because performance is heavily influenced by congestion, it is essential that a performance engineer be comfortable with quantitative analysis methods and have a solid grasp of basic statistics, queueing theory, and simulation methods. The wide variety of computer
Disciplines and Techniques Used in Performance Engineering

Technologies and the evolving set of problem domains mean that the performance engineer should have an eclectic set of skills and analysis methods at his or her disposal. In addition, it is useful for the performance engineer to know how to analyze large amounts of data with tools such as spreadsheets and scripting languages, because measurement data from a wide variety of sources may be encountered. Knowledge of statistical methods is useful for planning experiments and for understanding the limits of inferences that can be drawn from measurement data. Knowledge of queueing theory is useful for examining the limitations of design choices and the potential improvements that might be gained by changing them.

While elementary queueing theory may be used to identify limits on system capacity and to predict transaction loads at which response times will suddenly increase [DenningBuzen1978], more complex queueing theory may be required to examine the effects of service time variability, interarrival time variability, and various scheduling rules such as time slicing, preemptive priority, nonpreemptive priority, and cyclic service [Kleinrock1975, Kleinrock1976].

Complicated scheduling rules, load balancing heuristics, protocols, and other aspects of system design that are not susceptible to queueing analysis may be examined using approximate queueing models and/or discrete event simulations, whose outputs should be subjected to statistical analysis [LawKelton1982].

Queueing models can also be used in sizing tools to predict system performance and capacity under a variety of load scenarios, thus facilitating what-if analysis. This has been done with considerable commercial success. Also, queueing theory can be used to determine the maximum load to which a system should be subjected during performance tests once data from initial load test runs is available.

The performance engineer should have some grasp of computer science, software engineering, software development techniques, and programming so that he or she can quickly recognize the root causes of performance issues and negotiate design trade-offs between architects and developers when proposing remedies. A knowledge of hardware architectures, including processors, memory architectures, network technologies, and secondary storage technologies, and the ability to learn about new technologies as they emerge are very helpful to the performance engineer as well.

Finally, the performance engineer will be working with a wide variety of stakeholders. Interactions will be much more fruitful if the performance engineer is acquainted with the requirements drafting
and review processes, change management processes, architecture and design processes, and testing processes. The performance engineer should be prepared to work with product managers and business managers. He or she will need to explain choices and recommendations in terms that are related to the domain of application and to the trade-offs between costs and benefits.

1.6 Performance Modeling, Measurement, and Testing

Performance modeling can be used to predict the performance of a system at various times during its lifecycle. It can be used to characterize capacity; to help understand the impact of proposed changes, such as changes to scheduling rules, deployment scenarios, technologies, and traffic characteristics; or to predict the effect of adding or removing workloads. Deviations from the qualitative behavior predicted by queueing models, such as slowly increasing response times or memory occupancy when the system load is constant or expected to be constant, can be regarded as indications of anomalous system behavior. Performance engineers have used their understanding of performance models to identify software flaws; software bottlenecks, especially those occurring in new technologies that may not yet be well understood [ReeserHariharan2000]; system malfunctions (including the occurrence of deadlocks); traffic surges; and security violations. This has been done by examining performance measurement data, the results of simulations, and/or queueing models [AvBonWey2005, AvTanJaCoWey2010]. Interestingly, the principles that were used to gain insights into performance in these cases were independent of the technologies used in the system under study.

Performance models and statistical techniques for designing experiments can also be used to help us plan and interpret the results of performance tests.

An understanding of rudimentary queueing models will help us determine whether the measurement instrumentation is yielding valid values of performance metrics.

Pilot performance tests can be used to identify the ranges of transaction rates for which the system is likely to be lightly, moderately, or heavily loaded. Performance trends with respect to load are useful for predicting capacity and scalability. Performance tests at loads near or
above that at which any system resource is likely to be saturated will be of no value for predicting scalability or performance, though they can tell us whether the system is likely to crash when saturated, or whether the system will recover gracefully once the load is withdrawn. An understanding of rudimentary performance models will help us to design performance tests accordingly.

Methodical planning of experiments entails the identification of factors to be varied from one test run to the next. Fractional replication methods help the performance engineer to choose telling subsets of all possible combinations of parameter settings to minimize the number of experiments that must be done to predict performance.

Finally, the measurements obtained from performance tests can be used as the input parameters of sizing tools (based on performance models) that will assist in sizing and choosing the configurations needed to carry the anticipated load to meet performance requirements in a cost-effective manner.

1.7 Roles and Activities of a Performance Engineer

Like a systems architect, a performance engineer should be engaged in all stages of a software project. The performance engineer is frequently a liaison between various groups of stakeholders, including architects, designers, developers, testers, product management, product owners, quality engineers, domain experts, and users. The reasons for this are:

- The performance of a system affects its interaction with the domain.
- Performance is influenced by every aspect of information flow, including
  - The interactions between system components
  - The interactions between hardware elements and domain elements
  - The interactions between the user interface and all other parts of the system
  - The interactions between component interfaces

When performance and functional requirements are formulated, the performance engineer must ensure that performance and scalability requirements are written in verifiable, measurable terms, and that they are linked to business and engineering needs. At the architectural
stage, the performance engineer advises on the impacts of technology and design choices on performance and identifies impediments to smooth information flow. During design and development, the performance engineer should be available to advise on the performance characteristics and consequences of design choices and scheduling rules, indexing structures, query patterns, interactions between threads or between devices, and so on. During functional testing, including unit testing, the performance engineer should be alerted if the testers feel that the system is too slow. This can indicate a future performance problem, but it can also indicate that the system was not configured properly. For example, a misconfigured IP address could result in an indication by the protocol implementation that the targeted host is unresponsive or nonexistent, or in a failure of one part of the system to connect with another. It is not unusual for the performance engineer to be involved in diagnosing the causes of these problems, as well as problems that might appear in production.

The performance engineer should be closely involved in the planning and execution of performance tests and the interpretation of the results. He or she should also ensure that the performance instrumentation is collecting valid measurement data and generating valid loads. Moreover, it is the performance engineer who supervises the preparation of reports of performance tests and measurements in production, explains them to stakeholders, and mediates negotiations between stakeholders about necessary and possible modifications to improve performance.

If the performance of a system is found to be inadequate, whether in testing or in production, the performance engineer will be able to play a major role in diagnosing the technical cause of the problem. Using the measurement and testing methods described in this book, the performance engineer works with testers and architects to identify the nature of the cause of the problem and with developers to determine the most cost-effective way to fix it. Historically, the performance engineer’s first contact with a system has often been in “repairman mode” when system performance has reached a crisis point. It is preferable that performance issues be anticipated and avoided during the early stages of the software lifecycle.

The foregoing illustrates that the performance engineer is a performance advocate and conscience for the project, ensuring that performance needs are anticipated and accounted for at every stage of the development cycle, the earlier the better [Browne1981]. Performance advocacy includes the preparation of clear summaries of
performance status, making recommendations for changes, reporting on performance tests, and reporting on performance issues in production. Thus, the performance engineer should not be shy about blowing the whistle if a major performance problem is uncovered or anticipated. The performance reports should be concise, cogent, and pungent, because stakeholders such as managers, developers, architects, and product owners have little time to understand the message being communicated. Moreover, the performance engineer must ensure that graphs and tables tell a vivid and accurate story.

In the author’s experience, many stakeholders have little training or experience in quantitative methods unless they have worked in disciplines such as statistics, physics, chemistry, or econometrics before joining the computing profession. Moreover, computer science and technology curricula seldom require the completion of courses related to performance evaluation for graduation. This means that the performance engineer must frequently play the role of performance teacher while explaining performance considerations in terms that can be understood by those trained in other disciplines.

## 1.8 Interactions and Dependencies between Performance Engineering and Other Activities

Performance engineering is an iterative process involving interactions between multiple sets of stakeholders at many stages of the software lifecycle (see Figure 1.1). The functional requirements inform the specification of the performance requirements. Both influence the architecture and the choice of technology. Performance requirements may be formulated with the help of performance models. The models are used to plan performance tests to verify scalability and that performance requirements have been met. Performance models may also be used in the design of modifications. Data gathered through performance monitoring and capacity planning may be used to determine whether new functionality or load may be added to the system.

The performance engineer must frequently take responsibility for ensuring that these interactions take place. None of the activities and skills we have mentioned is sufficient for the practice of performance engineering in and of itself.
Figure 1.1 Interactions between performance engineering activities and other software lifecycle activities
1.9 A Road Map through the Book

Performance metrics are described in Chapter 2. One needs performance metrics to be able to define the desired performance characteristics of a system, and to describe the characteristics of the performance of an existing system. In the absence of metrics, the performance requirements of a system can be discussed only in vague terms, and the requirements cannot be specified, tested, or enforced.

Basic performance modeling and analysis are discussed in Chapter 3. We show how to establish upper bounds on system throughput and lower bounds on system response time given the amount of time it takes to do processing and I/O. We also show how rudimentary queueing models can be used to make predictions about system response time when a workload has the system to itself and when it is sharing the system with other workloads.

In Chapter 4 we explore methods of characterizing the workload of a system. We explain that workload characterization involves understanding what the system does, how often it is required to do it, why it is required to do it, and the performance implications of the nature of the domain of application and of variation in the workload over time.

Once the workload of the system has been identified and understood, we are in a position to identify performance requirements. The correct formulation of performance requirements is crucial to the choice of a sound, cost-effective architecture for the desired system. In Chapter 5 we describe the necessary attributes of performance requirements, including linkage to business and engineering needs, traceability, clarity, and the need to express requirements unambiguously in terms that are measurable, testable, and verifiable. These are preconditions for enforcement. Since performance requirements may be spelled out in contracts between a buyer and a supplier, enforceability is essential. If the quantities specified in a performance requirement cannot be measured, the requirement is deficient and unenforceable and should either be flagged as such or omitted. In Chapter 6 we discuss specific types of the ability of a system to sustain a given load, the metrics used to describe performance requirements, and performance requirements related to networking and to specific domains of application. In Chapter 7 we go into detail about how to express performance requirements clearly and how they can be managed.

One must be able to measure a system to see how it is functioning, to identify hardware and software bottlenecks, and to determine whether it is meeting performance requirements. In Chapter 8 we
describe performance measurement tools and instrumentation that can help one do this. Instrumentation that is native to the operating system measures resource usage (e.g., processor utilization and memory usage) and packet traffic through network ports. Tools are available to measure activity and resource usage of particular system components such as databases and web application servers. Application-level measurements and load drivers can be used to measure system response times. We also discuss measurement pitfalls, the identification of incorrect measurements, and procedures for conducting experiments in a manner that helps us learn about system performance in the most effective way.

Performance testing is discussed in Chapter 9. We show how performance test planning is linked to both performance requirements and performance modeling. We show how elementary performance modeling methods can be used to interpret performance test results and to identify system problems if the tests are suitably structured. Among the problems that can be identified are concurrent programming bugs, memory leaks, and software bottlenecks. We discuss suitable practices for the documentation of performance test plans and results, and for the organization of performance test data.

In Chapter 10 we use examples to illustrate the progression from system understanding to model formulation and validation. We look at cases in which the assumptions underlying a conventional performance model might deviate from the properties of the system of interest. We also look at the phases of a performance modeling study, from model formulation to validation and performance prediction.

Scalability is a desirable attribute of systems that is frequently mentioned in requirements without being defined. In the absence of definitions, the term is nothing but a buzzword that will engender confusion at best. In Chapter 11 we look in detail at ways of characterizing the scalability of a system in different dimensions, for instance, in terms of its ability to handle increased loads, called load scalability, or in terms of the ease or otherwise of expanding its structure, called structural scalability. In this chapter we also provide examples of cases in which scalability breaks down and discuss how it can be supported.

Intuition does not always lead to correct performance engineering decisions, because it may be based on misconceptions about what scheduling algorithms or the addition of multiple processors might contribute to system performance. This is the reason Chapter 12, which contains a discussion of performance engineering pitfalls, appears in
this book. In this chapter we will learn that priority scheduling does
not increase the processing capacity of a system. It can only reduce the
response times of jobs that are given higher priority than others and
hence reduce the times that these jobs hold resources. Doubling the
number of processors need not double processing capacity, because of
increased contention for the shared memory bus, the lock for the run
queue, and other system resources. In Chapter 12 we also explore pit-
falls in system measurement, performance requirements engineering,
and other performance-related topics.

The use of agile development processes in performance engineer-
ing is discussed in Chapter 13. We will explore how agile methods
might be used to develop a performance testing environment even if
agile methods have not been used in the development of the system as
a whole. We will also learn that performance engineering as part of an
agile process requires careful advance planning and the implementa-
tion of testing tools. This is because the time constraints imposed by
short sprints necessitate the ready availability of load drivers, measure-
ment tools, and data reduction tools.

In Chapter 14 we explore ways of learning, influencing, and telling
the performance story to different sets of stakeholders, including archi-
tects, product managers, business executives, and developers.

Finally, in Chapter 15 we point the reader to sources where more
can be learned about performance engineering and its evolution in
response to changing technologies.

1.10 Summary

Good performance is crucial to the success of a software system or a
system controlled by software. Poor performance can doom a system
to failure in the marketplace and, in the case of safety-related systems,
endanger life, the environment, or property. Performance engineering
practice contributes substantially to ensuring the performance of a
product and hence to the mitigation of the business risks associated
with software performance, especially when undertaken from the ear-
liest stages of the software lifecycle.
Index

A
ACID (Atomicity, consistency, isolation, and durability), 287
ACM (Association for Computing Machinery), 370
Agile software development
aligning tests with sprints, 329–330
communicating test results, 331
connection between irregular test results and incorrect functionality, 334
identifying and planning test and test instrumentation, 332–333
interpreting and applying test results, 330–331
methods for implementing tests, 332 overview, 325–327
performance engineering in, 327–328
performance requirements in, 328–329
playtime in testing process, 334–336
Scrum use in performance test implementation and performance test instrumentation, 333–334
summary and exercises, 336–337
Airport conveyor system example. see Conveyor systems, airport luggage example
Alarms. see Fire alarm system
Alerts, system measurement in triggering, 164
... “all the time/... of the time” antipattern, 145–146
Ambiguity
properties of performance requirements, 117–118
testing and, 158
Analysis. see Performance analysis
Antipatterns
... “all the time/... of the time” antipattern, 145–146
information flow review revealing, 347
number of users supported, 146–147
overview of, 144
performance antipattern (Smith and Williams), 300
resource utilization, 146
response time, 144–145
scalability, 147–148
Application domains, mapping to workloads
airport conveyor system example, 92–94
fire alarm system example, 94–95
online securities trading example, 91–92
overview, 91
Applications
processing time increasing per unit of work, 267
system measurement from within, 186–187
time-varying demand workload examples, 89–90
Architects
gathering performance requirements, 140
ownership of performance requirements, 156
stakeholder roles, 348–349
Architectural stage, of development, 12
Architecture
avoiding scalability pitfalls, 299
causes of performance failure, 4, 105–106
early testing to avoid poor choices, 113
hardware architectures, 9
performance engineering concerns influencing, 345–346
reviewing as step in performance engineering, 7
Architecture, continued
skills need by performance engineers, 8
structuring tests to reflect scalability of, 228–229
understanding before testing, 211–212
understanding impact of existing, 346–347
Arrival rate
characterizing queue performance, 42
connection between models, requirements, and tests, 79
formulating performance requirements to facilitate testing, 159
modeling principles, 201
quantifying device loadings and flow through computer systems, 56
Arrival Theorem (Sevcik-Mitrani Theorem), 70, 74
The Art of Computer Systems Performance Analysis (Jain), 371
Association for Computing Machinery (ACM), 370
Assumptions
in modeling asynchronous I/O, 262
in performance requirements documents, 152
Asynchronous activity
impact on performance bounds, 66–67
modeling asynchronous I/O, 260–266
parallelism and, 294
queueing models and, 255
Atomicity, consistency, isolation, and durability (ACID), 287
Audience, specifying in performance requirements document, 151–152
Automating
data analysis, 244–245
testing, 213, 244–245
Average device utilization
definition of common metrics, 20
formula for, 21
Average service time, in Utilization Law, 45–47
Average throughput, 20
Averaging time window, measuring utilization and, 175–177
B
Back-end databases, understanding architecture before testing, 211–212
Background activities
identifying concerns and drivers in performance story, 344–345
resource consumption by, 205
Bandwidth
linking performance requirements to engineering needs, 108
measuring utilization, 174–175
sustainable load and, 127
“Bang the system as hard as you can” testing method
equivalent way to evaluate throughput, 208–209
as provocative performance testing, 209–210
Banking systems
example of multiple-class queueing networks, 72
reference workload example, 88
scheduling periodic loads and peaks, 267
Baseline models
determining resource requirements, 7
using validated model as baseline, 255
Batch processing, in single-class closed queueing network model, 60
BCMP Theorem, 68, 73
Bentham, Jeremy, 20
Bohr bug, 209
Bottlenecks
contention and, 260
eliminating unmasks new pitfall, 319–321
improving load scalability, 294
measuring processor utilization by individual processes, 171
modeling principles, 201–202
performance modeling and, 10
in single-class closed queueing networks, 63
software bottlenecks, 314
upper bounds on system throughput and, 56–58
Bounds
- asymptotic bounds impacting throughput and response time, 63–66
- asynchronous activity impacting performance bounds, 66–67
- lower bounds impacting response time, 56–58
- upper bounds impacting response time, 129

Bugs, Bohr bug, 209

Business aspects
- linking performance requirements to needs, 108
- linking performance requirements to risk mitigation, 112–114
- of performance engineering, 6–7

Busy hour, measuring response time and transaction rates at, 26

Busy waiting, on locks, 285–286

Buyer-seller relationships, expertise and, 114–115

C
- C#, garbage collection and, 315
- CACM (Communications of the Association for Computing Machinery), 370

Calls, performance requirements related to lost calls, 134–135

Capacity management engineers, stakeholder roles, 355

Capacity planning
- applying performance laws to, 80
- creating capacity management plan, 167 measurement and, 165

Carried load, telephony metrics, 30–31

Carrier Sense Multiple Access with Collision Detection (CSMA/CD)
- bandwidth utilization and, 174
- load scalability and, 278

Central processing units. see CPUs

Central server model, simple queueing networks, 53–54

Central subsystem, computers, 61

Change management, skills need by performance engineers, 10

Checklists
- measurement, 192–193
- test, 219–220

Circuitous treasure hunt
- performance antipatterns and, 300–301
- review of information flow revealing, 347

Clocks. see System clocks

Closed queueing networks
- bottleneck analysis, 63
- defined, 59
- Mean Value Analysis, 69–71
- modeling asynchronous I/O and, 263
- qualitative view of, 62–63
- with single-class, 60–62

Clusters, of parallel servers, 195

CMG (Computer Measurement Group), 369

Coarse granularity locking,
- undermining scalability, 287

Code profiling, system measurement and, 190–191

Code segments, in measuring memory-related activity, 180

Collisions, load scalability and, 295

Commercial considerations
- buyer-seller relationships and, 114–115
- confidentiality, 115
- customer expectations and contracts and, 114
- outsourcing and, 116
- skills need by performance engineers and, 8

Commercial databases, measuring, 188–189

Communication, of test results in agile development, 331

Communications of the Association for Computing Machinery (CACM), 370

Competitive differentiators
- linking performance requirements to business needs, 108
- response time for web sites, 110

Completeness, of performance requirements, 119

Completion rate, in Utilization Law, 45–47

Compression, in achieving space scalability, 280

Computationally tractable, defined, 68
Computer Measurement Group (CMG), 369
Computer science, skills need by performance engineers, 9
Computer services, outsourcing, 116
Computer systems. see also Systems
  background activities in resource use, 205
  central subsystem, 61
  challenges posed by multiple-host systems, 218–219
  mapping application domains to workloads, 91–95
  modeling asynchronous I/O, 252–254
  quantifying device loadings and flow through, 54–56
  queueing examples, 38–39
  skills need by performance engineers and, 8
  system measurement and, 165–166
Computers and Operations Research, 370
Concurrency
  detecting/debugging issues, 223–224
  illusion of multiprocessing and, 54
  row-locking preferable to table-level locking, 301
Conferences, learning resources for performance engineering, 369–370
Confidentiality, commercial considerations related to performance requirements, 115
Confounding, undue cost of performance tests and, 22
Conservation, priority scheduling and, 311
Consistency
  ACID properties, 287
  mathematical consistency of performance requirements, 120, 148–149
  properties of metrics, 25
  workload specification and, 100
Contention. see Lock contention
Contracts
  between buyer and seller, 4
  commercial considerations related to performance requirements, 114
Control systems, understanding system architecture before testing, 211–212
Conveyor systems
  example of metrics applied to warehouse conveyor, 27–28
  example of time-varying demand workload, 89–90
  examples of background activities in resource use, 205
Conveyor systems, airport luggage
  example
  applying numerical data to workloads, 101–102
  mapping application domains to workloads, 92–94
  specifying workloads numerically, 97–98
  traffic patterns and, 99
Correctness, of performance requirements, 120
Costs
  of measurement, 182
  performance requirements and, 4–6
  of poor performance requirements, 113
  scalability and, 274
  traceability of performance requirements and, 121–122
CPUs. see also Processors
  benefits and pitfalls of priority scheduling, 310
  diminishing returns from multiprocessors or multiple cores, 320
  interpreting results of system with computationally intense transactions, 239–241
  interpreting results of system with memory leak and deadlocks, 242–243
  load scalability and scheduling rules and, 278–279
  measuring multicore and multiprocessor systems, 177–180
  measuring utilization, 21–22
  measuring utilization and averaging time window, 175–177
  playtime in testing process and, 335
  quantifying device loadings and flow through a computer system, 54–56
  resource utilization antipattern and, 146
simple queueing networks and, 53–54
single-server queues and, 42
sustainable load and, 127
time scale granularity in measuring utilization, 33
transient saturation not always bad, 312–314
utilization in service use cases example, 231–235
Crashes, stability and, 126
CSMA/CD (Carrier Sense Multiple Access with Collision Detection)
bandwidth utilization and, 174
load scalability and, 278
Custom relationship management, metrics example, 30–32
Customer expectations
as commercial consideration, 114
performance requirements and, 106–107

D
Data
collecting from performance tests, 229–230
reducing and interpreting in agile, 330–331
reducing and presenting, 230
in reporting performance status, 353–354
system measurement and organization of, 195
Data segments, 180
Database administrators, stakeholder roles, 353
Databases
background activities in resource use, 205
layout of performance requirements and, 153
measuring commercial, 188–189
parcel routing example, 258–260
understanding architecture before testing, 211–212
Deadlocks
benefit of testing of functional and performance requirements concurrently, 200
detecting/debugging, 224
implicit performance requirements and, 133
improving load scalability, 293, 295
interpreting results of system with memory leak and deadlocks, 241–243
measuring in commercial databases, 188–189
museum checkroom example, 289, 291
performance modeling and, 10
provocative performance testing, 210
verifying freedom from, 119
Decision making, understanding impact of prior decisions on system performance, 346–347
Dependencies
avoiding circular, 149–150
in performance engineering, 13–14
Derived performance requirements, 132
Design, negotiating design choices, 360–362
Designers
gathering performance requirements from, 140
as stakeholder role, 350–351
Developers
gathering performance requirements from, 140
as stakeholder role, 350–351
Development
agile development. see Agile software development
feature development, 164
model development, 254
software development. see Software development
waterfall development, 325
Development environment, scalability limits in, 292–293
Diagnosis, role of performance engineers, 12
Disciplines, in performance engineering, 8–10
Disjoint transactions, serial execution of, 283–285
Disk I/O, measuring utilization, 173. see also I/O (input/output)
Distance scalability, 281
Documents/documentation
performance requirements, 150–153
test plans and results, 220–222
Drafts
performance requirements, 7
tests based on performance
requirements drafts, 113
Durability property, ACID properties, 287

E
Ease of measurement, properties of
metrics, 25
Enforceability, of performance require-
ments, 143–144
Equilibrium
Markov chains and, 159, 231
of queue behavior, 50
Equipment, checking test equipment, 213–214
Erlang loss formula
applying to performance
requirements, 110
applying to probability of lost calls, 76–77
derived performance requirements
and, 132
Ethernet
comparing scalability with token
ring, 287–288
CSMA/CD and, 174
improving load scalability, 295
load scalability and, 278
Excel, statistical methods in, 374
Experimental plans, measurement
procedures, 192
Expert intent, in predicting
performance, 77
Expertise
buyer-seller relationships and, 114–115
investing in, 7
model sufficiency and, 255
Explicit metrics, 32
External performance requirements,
implications for subsystems, 150

F
Failure
interpreting results of transaction
system with high failure rate, 235–237
transaction failure rates, 134–135
Faults, system measurement in
detecting, 164
FCFS (First Come First Served)
regularity conditions for
computationally tractable
queueing network models, 68–69
types of queueing disciplines, 44
Feature set
in performance requirements
documents, 151
system measurement and feature
development, 164
Fields, in performance requirements
database, 154–155
Finite pool sizes, queues/queueing,
75–77
Fire alarm system
background activities impacting
resource use, 205
linking performance requirements to
regulatory needs, 110
mapping application domains to
system workloads, 94–95
metrics applied to, 28–29
occurrence of periodic loads and
peaks, 267
peak and transient loads and,
135–136
reference workload example, 88
specifying workloads numerically,
98–99
time-varying demand workloads in,
89–91
traffic pattern in, 99
First Come First Served (FCFS)
regularity conditions for computa-
tionally tractable queueing
network models, 68–69
types of queueing disciplines, 44
Flow balance, multiple-class queueing
networks and, 73
Forced Flow Law (Denning and Buzen)
benefits and pitfalls of priority
scheduling, 311
measurements conforming to, 168
modeling principles, 201–202
multiple-class queueing networks
and, 73
quantifying device loadings and
flows, 55
transaction loading and, 158
validating measurements, 191

Functional requirements
associating performance requirements with, 111
ensuring consistency of performance requirements with, 156
guidelines for, 116–117
performance requirements and, 117
referencing related requirements in performance requirements document, 152
specifying performance requirements and, 141–142
testing concurrently with performance requirements, 199–200

Functional testers, stakeholder roles, 351–352

Functional testing
executing performance test after, 327
performance requirements and, 106

G
Garbage collection
background activities in resource use, 205
performance engineering pitfalls, 315
Global response time (R₀), in single-class closed queueing network model, 61
gnuplot, plotting performance data, 218–219
“god class”
information flow review reveals antipatterns, 347
performance antipatterns and, 300
Goodput, telephony metrics, 30
Granularity
coarse granularity locking, 287
time scale in measuring utilization and, 32–33
time scale of performance requirements and, 122
Graphical presentation, in reporting performance status, 353–354

H
Head-of-the-line (HOL) priority, 311
Heisenberg Uncertainty Principle, 166, 209–210
HOL (head-of-the-line) priority, 311
Horizontal scaling (scaling out)
overview, 276–277
structuring tests to reflect scalability of architecture, 228
Hosts
measuring single- and multiple-host systems, 183–186
testing multiple-host systems, 218–219

I
ICPE (International Conference on Performance Engineering), 370
Idle counters, processor usage and, 169
IEEE Transactions on Software Engineering, 370
Implicit metrics, 32
Implicit performance requirements, 132–134
Income tax filing, electronic, example of time-varying demand workload, 90–91
Independence, properties of metrics, 25
Infinite loops, processor usage and, 169
Infinite Service (IS), regularity conditions for computationally tractable queueing network models, 68–69
Information, combining knowledge with controls, 94
Information flow, understanding impact of prior decisions on system performance, 347
Information processing, metrics example, 30–32
Input analysis, discrete event simulation, 373
Input/output (I/O). see I/O (input/output)
Instruments of measurement
aligning tests with sprints, 330
identifying and planning in agile development, 332–333
lagging behind software platforms and technologies, 340
overview, 166
Scrum use of, 333–334
scrutiny in use of, 168
validating, 168, 193–194
Integration tests
functional requirements and, 199–200
performance tests and, 202
Interactions
  in performance engineering, 13–14
  in performance requirements
documents, 151
Interarrival time, queueing and, 39–41.
  see also Arrival rate
International Conference on Perfor-
mance Engineering (ICPE), 370
Interoperability, in performance
requirements documents, 151
Interpreting measurements, in virtual
environments, 195
Interpreting test results
  applying results and, 330–331
  service use cases example, 231–235
  system with computationally intense
transactions, 237–241
  system with memory leak and
deadlocks, 241–243
  transaction system with high failure
rate, 235–237
Introduction to Queueing Theory
  (Cooper), 372
Investments, in performance
engineering, 6–7
I/O (input/output)
  asynchronous activity impacting
  performance bounds, 66–67
  benefits and pitfalls of priority
  scheduling, 310
  load scalability and scheduling rules
  and, 278–279
  measuring disk utilization, 173
  quantifying device loadings and flow
  through a computer system, 54–56
  single-server queues and, 42
  sustainable load and, 127
  where processing time increases per
  unit of work, 267
I/O devices
  modeling principles, 201
  in simple queueing networks,
  53–54
iostat (Linux/UNIX OSs), measuring
  CPU utilization, 171
IS (Infinite Service), regularity condi-
tions for computationally tractable
queueing network models,
  68–69
Isolation property, ACID properties, 287
J
Jackson’s Theorem
  multiple-class queueing networks
  and, 74
  single-class queueing networks and,
  59–60
Java
  garbage collection and, 315
  performance tuning resources, 374
  virtual machines, 317
Journal of the Association for Computing
  Machinery (JACM), 370
Journals, learning resources for perfor-
mance engineering, 369–370
K
Kernel mode (Linux/UNIX OSs),
  measuring CPU utilization, 171
L
Labs
  investing in lab time for measure-
  ment and testing, 7
  testing and lab discipline, 217
Last Come First Served (LCFS), 44
Last Come First Served Preemptive
  Resume (LCFSPR)
  regularity conditions for computa-
  tionally tractable queueing
  network models, 68–69
  types of queueing disciplines, 44
Layout, of performance requirements,
  153–155
Learning resources, for performance
  engineering
  conferences and journals, 369–370
  discrete event simulation,
  372–373
  overview, 367–369
  performance tuning, 374–375
  queueing theory, 372
  statistical methods, 374
  summary, 375
  system performance evaluation, 373
  texts on performance analysis,
  370–371
Legacy system, pitfall in transition to
  new system, 156–158
Linear regression, 374
Linearity, properties of metrics, 24–25
Index

Linux/UNIX OSs
- gathering host information, 185
- measuring memory-related activity, 180–181
- measuring processor utilization, 21–22, 170
- measuring processor utilization by individual processes, 172
- measuring processor utilization in server with two processors, 179
- skills need by performance engineers and, 8
- testing system stability, 225–226
- virtual machines mimicking, 316

LISP, garbage collection and, 315

Little's Law
- applying to processing time for I/O requests, 263
- connection between models, requirements, and tests, 79
- derived performance requirements and, 132
- Mean Value Analysis of single-class closed queueing networks, 71
- measurements conforming to, 168
- measurements from within applications and, 186
- modeling principles, 202
- overview, 47–49
- Response Time Law and, 61–62
- single-server queue and, 50
- verifying functionality of test equipment and software, 214

Live locks, detecting/debugging concurrency issues, 224

Load. see also Workloads
- deploying load drivers, 214–216
- museum checkroom example, 289
- occurrence of periodic, 267–268
- performance engineering addressing issues in, 5
- performance requirements in development of sound tests, 112
- performance requirements related to peak and transient loads, 135–136
- scalability. see Load scalability
- spikes or surges in, 91–92
- sustainable, 127–128
- systems with load-dependent behavior, 266

- telephony metrics example, 30–31
- testing background loads, 205
- testing load drivers, 214–216
- testing using virtual users, 190
- time-varying demand examples, 89–91

Load generation tools
- aligning tests with sprints, 330
- delayed time stamping as measurement pitfall, 319
- deploying software load drivers and, 214–216
- factors in choosing, 79
- incorrect approach to evaluating throughput, 208–209
- interpreting results of transaction system with high failure rate, 235–237
- measuring response time, 20–21
- planning performance tests, 203–204
- verifying functionality of test equipment, 213–214
- virtual users and, 190

Load scalability
- busy waiting on locks, 285–286
- coarse granularity locking, 287
- improving, 293–295
- interaction with structural scalability, 282
- limitations in a development environment, 292
- mathematical analysis of, 295–296
- overview, 277–279
- qualitative analysis of, 126, 283
- serial execution of disjoint transactions impeding, 283–285

Load tests
- aligning tests with sprints, 330
- background loads, 205
- load drivers and, 214–216
- performance requirements in development of sound tests, 112
- planning performance tests, 203–204
- virtual users in, 190

Lock contention
- bottlenecks and, 260
- busy waiting on locks, 285–286
- coarse granularity locking and, 287
- comparing implementation options for mutual exclusion, 296–298
- virtual machines and, 316
Locks
  benefits of priority scheduling for releasing, 309
  busy waiting, 285–286
  coarse granularity of, 287
  comparing with semaphores, 296–298
  row-locking vs. table-level locking, 301
Loops, processor usage and, 169
Lost calls/lost work
  performance requirements related to, 134–135
  queues/queueing and, 75–77
Lost packets, 134–135
Lower bounds, on system response times, 58

M
Management
  of performance requirements, 155–156
  as stakeholder, 349–350
Management information bases (MIBs), 185
Mapping application domains, to workloads
  example of airport conveyor system, 92–94
  example of fire alarm system, 94–95
  example of online securities trading system, 91–92
Market segments, linking performance requirements to size, 107–109
Markov chains, 159, 231
Markup language, modeling systems in development environment with, 292
Mathematical analysis, of load scalability, 295–296
Mathematical consistency
  ensuring conformity of performance requirements to performance laws, 148–149
  of performance requirements, 120
Mean service time, in characterizing queue performance, 42
Mean Value Analysis (MVA), of single-class closed queueing networks, 69–71
Measurability, of performance requirements, 118–119
Measurement
  collecting data from performance test, 229–230
  comparing with performance testing, 167–168
  investing in lab time and tools for, 7
  metrics applied to. see Metrics
  in performance engineering, 10–11
  performance engineering pitfalls, 317–319
  performance modeling and, 11
  performance requirements and, 118–119
  of systems. see System measurement
Measurement intervals, explaining to stakeholders, 356–359
Measurement phase, of modeling studies, 254
Measuring Computer Performance (Lilja), 371
Memory leaks
  interpreting measurements of system with memory leak and deadlocks, 241–243
  measuring from within applications, 186
  provocative performance testing and, 210
  sustainable load and, 127
  testing system stability, 225–226
Memory management
  background activities and, 205
  diminishing returns from multiprocessors or multiple cores, 320
  garbage collection causing degraded performance, 315
  measuring memory-related activity, 180–181
  performance engineering pitfalls, 321
  space-time scalability and, 280
Memory occupancy
  formulating performance requirements to facilitate testing, 159
  measuring memory-related activity, 180–181
  qualitative attributes of system performance, 126–127
Metrics, 23–24. see also Measurement; System measurement
  ambiguity and, 117–118
  applying to conveyor system, 27–28
applying to fire alarm system, 28–29
applying to information processing, 30–32
applying to systems with transient, bounded loads, 33–35
applying to telephony, 30
applying to train signaling and departure boards, 29–30
explicit and implicit, 32
focusing on single metric (mononumerosis), 26
gathering performance requirements and, 140
in numerical specification of workloads, 95
overview, 19–22
properties of, 24–26
reference workloads for domain-specific, 151
for scalability, 274–275
summary and exercises, 268–271
of systems with load-dependent or time-varying behavior, 266
understanding limits of, 251
Monitoring
airport conveyor system example, 93–94, 98
fire alarm system example, 95
online securities trading example, 101
in real-time systems, 317–318
Mononumerosis (tendency to focus on single metric), 26
mpstat (Linux/UNIX OSs)
measuring CPU utilization, 171
measuring processor utilization, 21, 284–285
measuring processor utilization in server with two processors, 179
Multicore systems
CPU utilization in, 170–171
detecting/debugging concurrency issues, 223–224
diminishing returns from, 314–315
measuring, 177–180
performance engineering concerns influencing architecture and technology choices, 346
Multiple-class queueing networks, 71–74
Multiple-host systems
challenges of testing, 218–219
measuring performance of, 183–186
Multiprocessor systems
CPU utilization in, 170–171
detecting/debugging concurrency issues, 223–224
diminishing returns from, 314–315
measuring, 177–180
performance engineering concerns influencing architecture and technology choices, 346
provocative performance testing, 210
Multitier configuration, of Web systems, 183–186
Munin tool, gathering measurements of multiple hosts with, 185
Museum checkroom, scalability example, 289–292, 298–299
Mutual exclusion, comparing semaphores with locks, 296–298

applying to systems with load-dependent or time-varying behavior, 266
understanding limits of, 251
Monitoring
airport conveyor system example, 93–94, 98
fire alarm system example, 95
online securities trading example, 101
in real-time systems, 317–318
Mononumerosis (tendency to focus on single metric), 26
mpstat (Linux/UNIX OSs)
measuring CPU utilization, 171
measuring processor utilization, 21, 284–285
measuring processor utilization in server with two processors, 179
Multicore systems
CPU utilization in, 170–171
detecting/debugging concurrency issues, 223–224
diminishing returns from, 314–315
measuring, 177–180
performance engineering concerns influencing architecture and technology choices, 346
Multiple-class queueing networks, 71–74
Multiple-host systems
challenges of testing, 218–219
measuring performance of, 183–186
Multiprocessor systems
CPU utilization in, 170–171
detecting/debugging concurrency issues, 223–224
diminishing returns from, 314–315
measuring, 177–180
performance engineering concerns influencing architecture and technology choices, 346
provocative performance testing, 210
Multitier configuration, of Web systems, 183–186
Munin tool, gathering measurements of multiple hosts with, 185
Museum checkroom, scalability example, 289–292, 298–299
Mutual exclusion, comparing semaphores with locks, 296–298

Index

applying to fire alarm system, 28–29
applying to information processing, 30–32
applying to systems with transient, bounded loads, 33–35
applying to telephony, 30
applying to train signaling and departure boards, 29–30
explicit and implicit, 32
focusing on single metric (mononumerosis), 26
gathering performance requirements and, 140
in numerical specification of workloads, 95
overview, 19–22
properties of, 24–26
reference workloads for domain-specific, 151
for scalability, 274–275
summary and exercises, 268–271
of systems with load-dependent or time-varying behavior, 266
understanding limits of, 251
Monitoring
airport conveyor system example, 93–94, 98
fire alarm system example, 95
online securities trading example, 101
in real-time systems, 317–318
Mononumerosis (tendency to focus on single metric), 26
mpstat (Linux/UNIX OSs)
measuring CPU utilization, 171
measuring processor utilization, 21, 284–285
measuring processor utilization in server with two processors, 179
Multicore systems
CPU utilization in, 170–171
detecting/debugging concurrency issues, 223–224
diminishing returns from, 314–315
measuring, 177–180
performance engineering concerns influencing architecture and technology choices, 346
Multiple-class queueing networks, 71–74
Multiple-host systems
challenges of testing, 218–219
measuring performance of, 183–186
Multiprocessor systems
CPU utilization in, 170–171
detecting/debugging concurrency issues, 223–224
diminishing returns from, 314–315
measuring, 177–180
performance engineering concerns influencing architecture and technology choices, 346
provocative performance testing, 210
Multitier configuration, of Web systems, 183–186
Munin tool, gathering measurements of multiple hosts with, 185
Museum checkroom, scalability example, 289–292, 298–299
Mutual exclusion, comparing semaphores with locks, 296–298

Index
MVA (Mean Value Analysis), of single-class closed queueing networks, 69–71

Number of users supported pattern/antipattern, 146–147

Numerical data, characterizing workloads with
airport conveyor system example, 101–102
fire alarm system example, 102–103
online securities trading example, 100–101
overview, 99

Numerical specification, of workloads
airport conveyor system example, 97–98
fire alarm system example, 98–99
online securities trading example, 96–97
overview, 95–96

Object pools
benefits of priority scheduling for the release of members of, 309
concurrent testing of functional and performance requirements and, 200
delayed time stamping as measurement pitfall, 319
finite pool sizes, 75–77
memory leaks and, 186
pool size requirement pattern, 147
validating measurements and, 192

Offered load
lost jobs and, 76
in telephony metrics, 30–31

One-lane bridge, performance antipatterns and, 300–301

One-step behavior, in Little's Law, 47

Online banking system. see also Banking systems
element of multiple-class queueing networks, 72
occurrence of periodic loads and peaks, 267

Online securities trading. see Securities trading example

Open queueing network models
defined, 59
modeling asynchronous I/O and, 263
qualitative view of queueing network representation, 62–63
single-class open queueing networks, 59–60

NDAs (nondisclosure agreements), confidentiality of performance requirements and, 115

Negotiation, regarding design choices and system improvement recommendations, 360–362

Network management systems (NMSs) applying metrics to systems with transient, bounded loads, 34–35
gathering host information with, 185
multiple-class queueing networks and, 72

Networks
scalability and congestion in, 281–282
scalability attribute of, 273
traffic in conveyor system model, 258

Networks of queues
applicability and limitations of simple queueing networks, 78
asymptotic bounds on throughput and response time, 63–66
asynchronous activity impacting performance bounds, 66–67
bottleneck analysis, 63
lower bounds on system response times, 58

Mean Value Analysis, 69–71
multiple-class queueing networks, 71–74
overview, 52–53
qualitative view of, 62–63
quantifying device loadings and flow, 54–56
regularity conditions for computationally tractable queueing network models, 68–69
simple queueing networks, 53–54
single-class closed queueing networks, 60–62
single-class open queueing networks, 59–60
upper bounds on system throughput, 56–58

Nondisclosure agreements (NDAs), confidentiality of performance requirements and, 115
Operating systems (OS)
- Linux/UNIX OSs. see Linux/UNIX OSs
- virtual machines mimicking, 316–317
- Windows OSs. see Windows OSs
Oracle
- commercial tools for measuring databases, 188
- performance tuning resources, 374
Organizational pitfalls, in performance engineering, 321–322
OS (operating systems). see Operating systems (OS)
Output analysis, discrete event simulation, 373
Outputs. see also I/O (input/output)
- automating analysis of, 244–245
- of conveyor system model, 258
Outsourcing, commercial considerations related to performance requirements, 116
Ownership
- ensuring of performance concerns, 360
- of performance requirements, 155–156

P
Packet handling
- examples of single-server queues and, 42
- performance requirements related to lost packets, 134–135
Packet-switched network, 135
Paged virtual memory systems, thrashing of, 266
Paging activity
- measuring memory-related activity, 181
- modeling principles and, 201
- thrashing of paged virtual memory systems, 266
PAL tool, in performance plot generation, 218–219
Parallelism
- improving load scalability and, 294
- interpreting measurements in virtual environments, 195
- load scalability undermined by inadequacy of, 279
- measuring parallel systems, 177–180
- multicore systems. see Multicore systems
- multiprocessors. see Multiprocessor systems
- single-threaded applications and, 314
Parameters, expressing performance requirements via, 149
Parcel routing database, 258–260
PASTA (Poisson Arrivals See Time Averages), 175
Patterns/antipatterns
- "all the time/... of the time" antipattern, 145–146
- information flow review revealing, 347
- number of users supported pattern/antipattern, 146–147
- overview of, 144
- pool size requirement pattern, 147
- resource utilization antipattern, 146
- response time pattern and antipattern, 144–145
- scalability antipattern, 147–148
Peak hour, measuring response time and transaction rates at, 26
Peak load
- issues addressed by performance engineering, 5
- occurrence of, 267–268
- performance requirements related to, 135–136
perfmon (Windows OSs)
- automating tests, 213
- measuring bandwidth utilization, 174
- measuring CPU utilization, 171
- measuring disk I/O, 173
- measuring memory-related activity, 180–181
- measuring processor utilization, 22
- measuring processor utilization by individual processes, 172
- measuring queue lengths, 175
- playtime in testing process in agile development and, 335
- testing system stability, 225–226
Performance analysis
- applying performance law to, 80
- asymptotic bounds and, 63–66
- of asynchronous activity impacting performance bounds, 66–67
Performance analysis, continued
of bottlenecks in single-class closed queueing networks, 63
finite pool sizes, lost calls, and lost work and, 75–77
investing in analysis tools, 7
of link between models, requirements, and tests, 79–80
Little’s Law in, 47–49
of lower bounds impact on system response times, 58
Mean Value Analysis of single-class closed queueing network models, 69–71
measurement procedures in, 194
of multiple-class queueing networks, 71–74
of networks of queues, 52–53
overview, 37
performance models in, 37–38
predicting performance based on, 77–78
qualitative view of queueing network representation, 62–63
quantifying device loadings and flow through a computer system, 54–56
of queueing causes, 39–41
of queueing in computer systems and in daily life, 38–39
of queueing performance, 42–45
of regularity conditions for computationally tractable queueing network models, 68–69
of simple queueing networks, 53–54, 78
of single-class closed queueing network model, 60–62
of single-class open queueing network model, 59–60
of single-server queue, 49–52
summary and exercises, 80–84
texts on performance analysis, 370–371
of upper bounds impact on system throughput, 56–58
Utilization Law and, 45–47
Performance antipattern (Smith and Williams), 144, 300. see also Antipatterns
Performance bounds. see Bounds
Performance engineering, introduction
business and process aspects of, 6–7
disciplines and techniques in, 8–10
example issues addressed by, 5–6
interactions and dependencies, 13–14
modeling, measuring, and testing, 10–11
overview, 1–4
performance requirements, 4–5
road map to topics covered in book, 15–17
roles/activities of performance engineers, 11–13
summary, 17
Performance engineering pitfalls. see Pitfalls
Performance engineers
lead role in performance requirements gathering, 141
as owner of performance requirements, 156
roles/activities of, 11–13
system measurement by, 163
Performance Evaluation Review, 370
Performance laws
ensuring conformity of performance requirements to, 148–149
Little’s Law. see Little’s Law
Utilization Law. see Utilization Law
Performance metrics. see Metrics
Performance Modeling and Design of Computer Systems: Queueing Theory in Action (Harchol-Balter), 371
Performance models. see Models
Performance requirements
in agile development, 328–329
... “all the time/... of the time” antipattern, 145–146
avoiding circular dependencies, 149–150
business risk mitigation and, 112–114
commercial considerations, 114–116
completeness of, 119
complying with regulatory needs, 108–110
concurrent testing of functional requirements, 199–200
conforming to performance laws, 148–149
connection between models, requirements, and tests, 79–80
consistency of, 120
correctness of, 120
derived, 132
drafting, 7
eliciting and gathering, 140–143
eliciting, writing, and managing, 139
ensuring enforceability of, 143–144
expressing in terms of parameters
  with unknown values, 149
formulating generally, 253
formulating response time
  requirements, 128–130
formulating throughput
  requirements, 130–131
formulating to facilitate testing,
  158–160
functional requirements and, 117
granularity and time scale of, 122
guidelines for specifying, 116–117
implications of external requirements
  for subsystems, 150
implicit, 132–134
layout of, 153–155
linking tests to, 222–223
managing, 155–156
measurability of, 118–119
meeting workload size, 107–108
number of users supported pattern/
  antipattern, 146–147
overview, 105–106, 125–126
patterns/antipatterns. see Patterns/
  antipatterns
performance engineering pitfalls and,
  321
pool size requirement pattern, 147
product management and, 106–107
qualitative attributes of system
  performance, 126–127
questions to ask in determining, 86–87
related to peak and transient loads,
  135–136
related to transaction failure rates,
  lost calls, lost packets, 134–135
resource utilization antipattern, 146
response time pattern and
  antipattern, 144–145
role in performance engineering, 4–5
scalability antipattern, 147–148
software lifecycle and, 111–112
storing and reporting, 160–161
structuring documentation of, 150–153
summary and exercises, 122–124,
  136–138, 161
supporting revenue streams, 110
sustainable load and, 127–128
testability of, 120–121
traceability of, 121–122
transitioning from legacy system
  and, 156–158
unambiguous quality of, 117–118
Performance Solutions (Smith and
  Williams), 371
Performance story
determining which performance
  aspects matter to stakeholders,
  340–341
e nsuring ownership of performance
  concerns, 360
explaining measurement intervals to
  stakeholders, 356–359
identifying concerns, drivers, and
  stakeholders, 344–345
most pressing questions in, 343–344
negotiating design choices and
  system improvement recommenda-
  tions, 360–362
overview, 339–340
reporting performance status to
  stakeholders, 353–354
sharing/developing with
  stakeholders, 347–348
stakeholder influence on, 345
understanding impact of existing
  architecture, 346–347
using performance engineering
  concerns to influence architecture
  and technology choices, 345–346
where it begins, 341–343
Performance test plan
documenting, 220–222
linking tests to performance
  requirements in, 222–223
measurement procedures in, 193
overview, 4
system measurement and, 168
system stability and, 225–226
Performance testers, stakeholder roles,
  351–352
Performance tests
applying performance laws to, 80
automating, 213, 244–245
background loads and, 205
Performance tests, continued
basing on performance requirement draft, 113
challenges in, 202–203
checking test equipment and software, 213–214
collecting data from, 229–230
comparing performance measurement with performance testing, 167–168
comparing production measurement with performance testing and scalability measurement, 181–183
connection between models, requirements, and tests, 79–80
costs of, 22
deploying load drivers, 214–216
detecting/debugging concurrency issues, 223–224
developing sound tests, 112
documenting plans and results, 220–222
evaluating linearity of utilization, 205–208
element of wrong way to evaluate throughput, 208–209
formulating performance requirements to facilitate, 158–160
interpreting results of service use cases example, 231–235
interpreting results of system with computationally intense transactions, 237–241
interpreting results of system with memory leak and deadlocks, 241–243
interpreting results of transaction system with high failure rate, 235–237
lab discipline in, 217
linking performance requirements to size, 108
linking tests to performance requirements, 222–223
measurement procedures, 193–194
measuring multiprocessor systems, 179–180
modeling and, 10
multiple-host systems and, 218–219
overview of, 199–202
in performance engineering, 10–11
performance test plan, 168
planning tests and models, 203–204
planning tests for system stability, 225–226
preparing for, 210–211
provocative performance testing, 209–210
reducing and presenting data, 230
regulatory and security issues, 216–217
role of performance engineers in, 12
scalability and, 302–303
scripts and checklists, 219–220
steps in performance engineering, 7
structuring to reflect scalability of architecture, 228–229
summary and exercises, 246–249
understanding system architecture and, 211–212
unspecified requirements and (prospective testing), 226–227
Performance tests, in agile development
aligning with sprints, 329–330
communicating test results, 331
identifying and planning tests and test instrumentation, 332–333
implementing, 332
interpreting and applying test results, 330–331
link between irregular test results and incorrect functionality, 334
playtime in testing process, 334–336
Scrum use in test implementation and test instrumentation, 333–334
Performance tuning, learning resources for, 374–375
pidstat (Linux/UNIX OSs), 225–226
Pilot tests
automating, 213
measuring multiprocessor systems, 179–180
performance modeling and, 10–11
ping command, determining remote node is operational, 34
Pitfalls
diminishing returns from multiprocessors or multiple cores, 314–315
eliminating bottleneck un masks new pitfall, 319–321
Index

garbage collection, 315
measurement and, 317–319
organizational, 321–322
overview, 307–308
priority scheduling, 308–312
scalability, 299–302
summary and exercises, 322–323
transient CPU saturation, 312–314
in transition from legacy system to
new system, 156–158
virtual machines and, 315–317
Plans/planning
capacity planning, see Capacity planning
documenting, 220–222
identifying, 332–333
models and, 203–204
performance test planning, see
Performance test plan
Platforms, system measurement and,
164
Playbook, creating scripts and checklists
for performance testing, 219–220
Playtime
automating tests, 213
in testing process, 334–336
PLCs (programmable logic controllers),
256–258
Plotting tools, automating plots of
performance data, 218–219
Poisson arrivals, 74
Poisson Arrivals See Time Averages
(PASTA), 175
Pool size, see also Object pools
finite pool sizes, 75–77
pool size requirement pattern, 147
The Practical Performance Analyst
(Gunther), 370
Predicting performance, 77–78
Preemptive priority, 311
Priority scheduling, see also Scheduling rules
benefits and pitfalls, 347
preemptive priority, 311
Privacy, regulations in performance tests
and, 217
Procedures, system measurement,
192–194
Process aspects, of performance
engineering, 6–7
Processes
scalability attribute of, 273
synchronization on virtual machines, 316
Processor Sharing (PS)
regularity conditions for computa-
tionally tractable queueing
network models, 68–69
types of queueing disciplines, 45
Processors, see also CPUs
diminishing returns from multipro-
cessors or multiple cores, 314–315
measuring multicore and multipro-
cessor systems, 177–180
measuring processor utilization,
21–22, 169–171
measuring processor utilization and
averaging time window, 175–177
measuring processor utilization by
individual processes, 171–173
modeling principles, 201
quantifying device loadings and flow
through a computer system. see
CPUs
Product form, 59
Product management, performance
requirements and, 106–107
Product managers, gathering perfor-
mance requirements from, 141
Production
comparing production measurement
with performance testing and
scalability measurement,
181–183
system measurement in production
systems, 164
Programmable logic controllers (PLCs),
256–258
Programming, skills need by perfor-
mance engineers, 9
Projection phase, phases of modeling
studies, 254–255
Properties
ACID properties, 287
of performance metrics, 24–26,
191–192
of performance requirements,
117–118
Prospective testing, when requirements
are unspecified, 226–227
Provocative performance testing, 209–210
PS (Processor Sharing)
  regularity conditions for computationally tractable queueing network models, 68–69
types of queueing disciplines, 45
ps command (Linux/UNIX OSs)
  measuring memory-related activity, 180–181
  obtaining processor utilization, 284–285
testing system stability, 225–226
Pure delay servers, 61
Q
Qualitative attributes, of system performance, 126–127
Qualitative view, of queueing network representation, 62–63
Quality of service (QoS), linking performance requirements to, 110
Quantitative analysis, skills need by performance engineers, 8
Quantitative System Performance (Lazowska et al.), 371
Queries, performance issues in databases, 188
Queue length
  for airport luggage workload, 93
  in characterizing queue performance, 43
  connection between models, requirements, and tests, 79
  in measuring utilization, 175
  single-server queue and, 51
Queueing models/theory
  model sufficiency and, 255
  modeling asynchronous I/O and, 263
  performance modeling and, 10
  skills need by performance engineers, 8–9
Queueing Systems, Volume 1: Theory (Kleinrock), 372
Queueing Systems, Volume 2: Applications (Kleinrock), 372
Queues (Cox and Smith), 372
Queueing asymptotic bounds on throughput and response time, 63–66
asynchronous activity impacting performance bounds, 66–67
avoiding scalability pitfalls, 301–302
bottleneck analysis, 63
causes of, 39–41
characterizing performance of, 42–44
in computer systems and in daily life, 38–39
connection between models, requirements, and tests, 79–80
finite pool sizes, lost calls, and lost work, 75–77
learning resources for, 372
limitations/applicability of simple models, 78
Little’s Law and, 47–49
load scalability and, 279
lower bounds on system response times, 58
Mean Value Analysis, 69–71
measuring queueing time in multicore and multiprocessor systems, 177–180
multiple-class queueing networks, 71–74
museum checkroom example, 289–290
networks of queues, 52–53
predicting performance and, 77–78
product form and, 59
priority scheduling and, 311, 347
qualitative view of, 62–63
quantifying device loadings and flow through a computer system, 54–56
regularity conditions for computationally tractable queueing network models, 68–69
simple queueing networks, 53–54
single-class closed queueing network model, 60–62
single-class open queueing network model, 59–60
single-server queue, 49–52
types of queueing disciplines, 44–45
upper bounds on system throughput, 56–58
Utilization Law and, 45–47
Index 403

R programming language, statistical methods and, 374

$R_0$ (global response time), in single-class closed queueing network model, 61

RAID devices, modeling asynchronous I/O and, 265–266

Railway example, linking performance requirements to regulatory needs, 109

Real-time systems, monitoring in, 317–318

Reducing data, 230, 330–331

Reference workloads
  in performance requirements documents, 151
  for systems with differing environments, 87–88

Regression analysis
  obtaining service demand, 254
  statistical methods and, 374

Regularity conditions, for computationally tractable queueing network models, 68–69

Regulations
  financial regulations in performance tests, 216–217
  linking performance requirements to, 108–110
  in performance requirements documents, 152

Reliability
  under load, 4
  properties of metrics, 25

Repeatability, properties of metrics, 25

Reports/reporting
  investing in reporting tools, 7
  online securities trading example, 96
  performance requirements and, 160–161
  performance status to stakeholders, 353–354
  role of performance engineers in, 12–13

Requirements
  functional requirements. see Functional requirements
  performance requirements. see Performance requirements
  Requirements engineers, stakeholder roles, 350

Resident set, in Linux/UNIX systems, 180

Resources
  background activities in resource use, 205
  determining resource requirements, 7
  measuring utilization, 158–159, 168–169
  priority scheduling not always beneficial or cost-effective, 308
  resource utilization antipattern, 146
  user experience metrics vs. resource metrics, 23–24

Resources, for learning. see Learning resources, for performance engineering

Response time
  airport luggage workload, 93
  asymptotic bounds on, 63–66
  asynchronous activity impacting, 66–67
  asynchronous I/O and, 260–266
  challenges in testing systems with multiple hosts, 218–219
  in characterizing queue performance, 42–43
  common metrics for, 20
  as competitive differentiator for web sites, 110
  connection between models, requirements, and tests, 79
  delayed time stamping as measurement pitfall, 317–318
  detecting/debugging concurrency issues in multiprocessor systems, 223–224
  facilitating testing, 159
  formula for average response time, 20–21
  formulating response time requirements, 128–130
  global response time ($R_0$) in single-class queueing model, 61
  in Little’s Law, 47–49
  lower bounds on system response times, 56–58
  measuring at peak or busy hour, 26
  measuring generally, 189–190
Response time, continued
- measuring in commercial databases, 188
- modeling principles, 201
- pattern and antipattern, 144–145
- pitfalls, 321
- qualitative view of queueing networks, 62–63
- response time pattern and antipattern, 144–145
- single-server queue and, 51
- sustainable load and, 128
- system design and, 6
- unbiased estimator of variance of, 22
- upper bounds on, 129
- validating measurements and, 192
- validation phase of model and, 254

Response Time Law
- applying to capacity planning and performance testing, 80
- combining inequality with, 65–66
- connection between models, requirements, and tests, 79
- delayed time stamping as measurement pitfall, 319
- relating think time, average response time, and system throughput, 61–62

Response time pattern and antipattern, 144–145

Revenue streams, linking performance requirements to, 110

Review process, skills need by performance engineers, 10

Risks
- performance and, 6
- reducing business risk, 112–114

Road traffic control system, 88

Road map, to topics covered in this book, 15–17

Round Robin, types of queueing disciplines, 44

Round-trip times, linking performance requirements to engineering needs, 108

S
- S programming language, statistical methods, 374
- Safety checks, testing and, 193–194

Sample statistics, comparing with time-average statistics, 21

sar (Linux/UNIX OSs)
- measuring processor utilization, 22, 171
- measuring processor utilization by individual processes, 172–173
- testing system stability, 225–226

Sarbanes-Oxley financial regulations, in performance tests, 216

Saturation
- diminishing returns from multiprocessors or multiple cores, 320
- equilibrium and, 50
- transient saturation not always bad, 312–314
- utilization and, 56

Utilization Law and, 45–46

Saturation epoch, 312–313

Scalability
- antipattern, 147–148
- avoiding pitfalls of, 299–302
- busy waiting on locks and, 285–286
- causes of system performance failure, 106–107
- coarse granularity locking and, 287
- comparing options for mutual exclusion, 296–298
- comparing production measurement with performance testing and scalability measurement, 181–183
- definitions of, 275
- in Ethernet/token ring comparison, 287–288
- improving load scalability, 293–295
- interactions between types of, 282
- limitations in development environment, 292–293
- load scalability, 277–279
- mathematical analysis of, 295–296
- museum checkroom example, 289–292, 298–299
- over long distances and network congestion, 281–282
- overview, 273–275
- performance tests and, 302–303
- qualitative analysis of, 283
- qualitative attributes of system performance, 126
- scaling methods, 275
serial execution of disjoint transactions, 283–285
space scalability, 279–280
space-time scalability, 280–281
structural scalability, 281
structuring tests to reflect, 228–229
summary and exercises, 303–305
types of, 277
Scalability antipattern, 147–148
Scaling methods, 275
Scaling out (horizontal scaling) overview, 276–277
structuring tests to reflect scalability of architecture, 228
Scaling up (vertical scaling) overview, 276–277
structuring tests to reflect scalability of architecture, 228
Scheduling
aligning tests with sprints, 329–330
periodic loads and peaks, 267–268
Scheduling rules
avoiding scalability pitfalls, 299
improving scalability and, 294
load scalability and, 278
not always improving performance, 308
pitfalls related to priority scheduling, 308–312
qualitative view of queueing networks, 62–63
Scope and purpose section, in performance requirements documents, 151
Scripts
creating test scripts, 219–220
for verifying functionality, 213
Scrums, in agile test implementation and instrumentation, 333–334
Scrutiny, in system measurement, 168
Securities trading example
applying numerical data to characterize workloads, 100–101
linking performance requirements to, 110
mapping application domains to system workloads, 91–92
numerical specification of workloads, 96–97
time-varying demand on workload, 89–90
traffic patterns and, 99
Security, testing financial systems and, 216–217
Seller-buyer relationships, performance expertise and, 114–115
Semaphores
comparing with locks, 296–298
load scalability and, 294–295
mathematical analysis of load scalability, 295–296
Sensitivity analysis, 78
Serial execution, of disjoint transactions, 283–285
Servers, central server model, 53–54
Service rate, queueing and, 41
Service time
connection between models, requirements, and tests, 79
in Little’s Law, 49
qualitative view of queueing networks, 62–63
queueing and, 39–41
single-server queue and, 52
in Utilization Law, 45–47
Services
obtaining service demand, 254
outsourcing, 116
use cases, 231–235
SIGMETRICS (ACM Special Interest Group on Performance Evaluation), 370
SIGSOFT (ACM Special Interest Group on Software Engineering), 370
Simple Network Management Protocol (SNMP), 185
Simple queueing networks. see Networks of queues
Simulation
discrete event simulation, 372–373
skills need by performance engineers, 8
Single-class closed queueing networks
asymptotic bounds on throughput and response time, 63–66
asynchronous activity impacting performance bounds, 66–67
bottleneck analysis, 63
Mean Value Analysis, 69–71
Index

Single-class open queueing networks, 59–62
Single-host systems, measuring performance of, 183
Single-server queue, 49–52
Size, linking performance requirements to, 107–108
SNMP (Simple Network Management Protocol), 185
Software
aligning tests with sprints, 329–330
associating performance requirements with lifecycle of, 111–112
checking test equipment and, 213–214
deploying load drivers, 214–216
examples of importance of performance in systems, 2–3
performance requirements and, 106
Software bottleneck
diminishing returns from multiprocessors or multiple cores, 314–315
eliminating bottleneck unmasks new pitfall, 320
Software development
agile approach. see Agile software development
outsourcing, 116
skills need by performance engineers, 9
waterfall approach, 325
Software development cycle
interactions and dependencies in performance engineering and, 14
limitation of “build it, then tune it” approach, 3–4
performance requirements and, 155
Software engineering, skills need by performance engineers, 9
Software project failure
interpreting results of transaction system with high failure rate, 235–237
transaction failure rates, 134–135
Sojourn time
in characterizing queue performance, 42–43
Space dimension, sustainable load and, 128
Space scalability, 126, 279–280
Space-time scalability, 280–281, 292
SPEC (Standard Performance Evaluation Corporation), 369
SPEC Benchmark Workshop (2009), 371
Special Interest Group on Performance Evaluation (SIGMETRICS), 370
Special Interest Group on Software Engineering (SIGSOFT), 370
Specification
benefits of performance requirements, 113
of functional and performance requirements, 141–142
guidelines for, 116–117
of workloads. see Numerical specification, of workloads
Speed/distance scalability, 281–282
Spreadsheet packages, 374
Sprints (iterations)
in agile development, 325
aligning tests with, 329–330
identifying and planning tests and test instrumentation, 332–333
performance requirements evolving between, 328
SQL, commercial tools for measuring databases, 188
Stability
planning tests for system stability, 225–226
qualitative attributes of system performance, 126
system measurement and, 165
Stakeholders
architects, 348–349
capacity management engineers, 355
designers and developers, 350–351
determining which performance aspects matter to, 340–341
ensuring ownership of performance concerns, 360
example of working with, 354–355
explaining concerns and sharing performance story with, 347–348
explaining measurement intervals to, 356–359
facilitating access to performance requirements, 155
functional testers and performance testers, 351–352
identifying performance concerns, drivers, and stakeholders, 344–345
influencing performance story, 345
interactions and dependencies and, 13
model sufficiency and, 255
negotiating design choices and
system improvement recommenda-
tions, 360–362
overview, 339–340
performance engineers relating to, 13
in performance requirements
documents, 151–152
performance story and, 341–344
relationship skills need by perfor-
mance engineers, 9–10
reporting performance status to,
353–354
requirements engineers and, 350
requirements gathering and, 140
roles of, 349–350
summary and exercises, 362–366
system administrators and database
administrators, 353
testers, 351–352
understanding impact of existing
architecture on system perfor-
mance, 346–347
user experience engineers, 352–353
using performance engineering
corns to influence architecture
and technology choices, 345–346
Standard Performance Evaluation
Corporation (SPEC), 369
Standards, citation of and compliance
with, in performance require-
ments documents, 152
State transition diagrams, 287–288
Statistics
comparing time-average with
sample, 21
learning resources for statistical
evaluation, 374
performance modeling and, 10
skills need by performance engi-
neers, 8–9
System administrators, 141
System architecture. see also
Architecture
reviewing as step in performance
engineering, 7
skills need by performance engi-
neers, 8
testing and, 211–212
System clocks
clock drift causing measurement
errors, 317
synchronization in multiple-host
systems, 184
System configuration
provocative performance testing, 210
understanding system architecture
before testing, 212
System managers, system measurement
by, 163
System measurement. see also
Measurement; Metrics
from within applications, 186–187
bandwidth utilization, 174–175
code profiling, 190–191
of commercial databases, 188–189
comparing measurement with
testing, 167–168
comparing production measurement
with performance testing and
scalability measurement, 181–183
data organization and, 195
disk utilization, 173
interpreting measurements in virtual
environments, 195
memory-related activity, 180–181
in middleware, 187–188
multicore and multiprocessor
systems, 177–180
overview of, 163–167
procedures, 192–194
processor usage, 169–171
processor usage by individual
processes, 171–173
queue length, 175
System measurement, continued
resource usage, 168–169
response time, 189–190
of single-host and multiple-host
systems, 183–186
summary and exercises, 196–197
utilizations and the averaging time
window, 175–177
validating with basic properties of
performance metrics, 191–192
validation and scrutiny in, 168
System mode (Linux/UNIX OSs), 171
System resources, 308. see also Resources
Systems
application where processing time
increases per unit of work over
time, 267
conveyor system example, 256–260
interpreting results of system with
computationally intense transac-
tions, 237–241
learning resources for evaluating
performance of, 373
load scalability and, 279
with load-dependent or time-varying
behavior, 266
lower bounds on response time, 58
measuring response time, 189
modeling asynchronous I/O, 260–266
modeling computer systems, 252–254
negotiating system improvement
recommendations, 360–362
phases of modeling studies, 254–256
pitfall in transition from legacy
system, 156–158
planning tests for stability of,
225–226
qualitative attributes of performance,
126–127
reference workloads in different
environments, 87–88
scalability antipattern, 147–148
scalability attribute of, 273–275
scheduling periodic loads and peaks,
267–268
summary and exercises, 268–271
thrashing of paged virtual memory
systems, 266
with transient, bounded loads, 33–35
understanding, 251
understanding impact of existing
architecture on, 346–347
upper bounds on throughput, 56–58
T
Task Manager (Window OSs)
automating tests, 213
interpreting results of system with
memory leak and deadlocks, 243
measuring CPU utilization, 171
measuring memory-related activity,
181
testing system stability, 225–226
Tax filing, occurrence of periodic loads
and peaks, 267–268
TCP/IP, speed/distance scalability of,
281–282
Techniques, in performance engineering,
8–10
Technologies
evaluating performance
characteristics of, 7
using performance engineering
concerns to influence, 345–346
Telephone call center
performance requirements related to
transaction failure rates, lost calls,
lost packets, 134–135
time-varying demand workload
examples, 89–90
Telephony
background activities in resource use,
205
implicit performance requirements
and, 133
lost jobs and, 75
metrics example, 30
structural scalability and, 281
Test plan. see Performance test plan
Testability, of performance require-
ments, 120–121
Testers, gathering performance require-
ments from, 140
Tests
functional. see Functional testing
integration. see Integration tests
performance. see Performance tests
pilot. see Pilot tests
unit. see Unit tests
Texts, on performance analysis, 370–371
Think time, in single-class closed queueing network model, 61
Thrashing, of paged virtual memory systems, 266
Thread safety
  concurrent testing of functional and performance requirements, 200
  detecting/debugging concurrency issues, 223–224
Threads, synchronization on virtual machines, 316
Throughput
  applying performance law to, 80
  asymptotic bounds on, 63–66
  characterizing queue performance, 42
  detecting/debugging concurrency issues, 224
  in Ethernet/token ring comparison, 287–288
  example of wrong way to evaluate, 208–209
  formulating throughput requirements, 130–131
  in Little’s Law, 48
  lower bounds on, 58
  modeling principles, 201
  pitfalls, 321
  quantifying device loadings and flow through a computer system, 55
  replacing coarse granularity locks with fine-grained locks, 287
  speed/distance scalability and, 282
  telephony metrics, 30–31
  upper bounds on, 56–58
Time (temporal) dimension, sustainable load and, 128
Time scalability, 126
Time scale
  granularity and time scale of performance requirements, 122
  metrics and, 32–33
Time slicing, types of queueing disciplines, 44
Time stamps, delay as measurement pitfall, 317–318
Time-average statistics, 21
Time-varying behavior
  systems with, 266
  workloads and, 88–91
Token ring
  comparing scalability with Ethernet, 287–288
  improving load scalability, 295
Traceability, of performance requirements, 121–122
Traffic intensity
  in characterizing queue performance, 43
  single-server queue, 49
Train signaling and departure boards, metrics example, 29–30
Transaction failure rates. see also Failure
  interpreting results of transaction system with high failure rate, 235–237
  performance requirements related to, 134–135
Transaction rates
  ambiguity and, 117–118
  completeness of performance requirements and, 119
  evaluating linearity of utilization with respect to transaction rate, 205–208
  measuring at peak or busy hour, 26
  measuring from within applications, 186
  online securities trading example, 96
  validating measurement of, 191
Transaction-oriented systems, understanding system architecture before testing, 211–212
Transactions
  interpreting results of system with computationally intense transactions, 237–241
  interpreting results of transaction system with high failure rate, 235–237
  serial execution of disjoint transactions, 283–285
Transient load, performance requirements related to, 135–136
U
Unambiguousness, properties of performance requirements, 117–118
Unit tests
  functional requirements and, 199–200
Unit tests, continued
role of performance engineers, 12
verifying functionality with, 189
UNIX OS. see Linux/UNIX OSs
Upper bounds, on system throughput, 56–58
Use cases, interpreting results of service use cases, 231–235
User base
number of users supported pattern/antipattern, 146–147
performance impact of increasing size of, 5–6
User experience engineers, stakeholder roles, 352–353
User experience metrics, 23–24
Utilization (U)
applying performance law to, 80
characterizing queue performance, 43
connection between models, requirements, and tests, 79
CPU utilization in service use cases example, 231–235
evaluating linearity with respect to transaction rate, 205–208
interpreting results of system with computationally intense transactions, 239–241
measuring bandwidth utilization, 174–175
measuring processor utilization, 169–171
measuring processor utilization and averaging time window, 175–177
measuring processor utilization by individual processes, 172–173
modeling principles, 201
quantifying device loadings and flow through a computer system, 55–56
resource utilization antipattern, 146
sustainable load and, 127–128
synchronous and asynchronous activity and, 263
in Utilization Law, 45–47
validating measurements and, 191
Utilization Law
applying to capacity planning and performance testing, 80
connection between models, requirements, and tests, 79
derived performance requirements and, 132
measurements conforming to, 168
measuring utilization in server with two processors, 179
modeling principles, 201
obtaining service demand, 254
overview, 45–47
performance test planning and, 204
resource utilization measurement, 169
statistical methods and, 374
transaction loading and, 158
Validation
predicting performance and, 78
of system measurement, 168
of system measurement with basic properties of performance metrics, 191–192
Validation phase, of modeling studies, 254
Verifiability, of performance requirements, 118–119
Vertical scaling (scaling up)
overview, 276–277
structuring tests to reflect scalability of architecture, 228
Virtual clients, 236
Virtual environments, interpreting measurements in, 195
Virtual machines, performance engineering pitfalls and, 315–317
Virtual users, load testing with, 190
Visit ratios, CPUs, 54
vmstat (Linux/UNIX OSs)
measuring CPU utilization, 171
measuring memory-related activity, 181
testing system stability, 225–226
Waiting time
in characterizing queue performance, 43
priority scheduling and, 309
Waterfall development, 325
Web sites, implicit performance requirements and, 133–134
Web systems, multiplier configuration of, 183–186
Web-based online banking system, example of multiple-class queueing networks, 72
What-if-analysis
predicting performance and, 78
projecting changes, 255
Windows OSs
automating tests, 213
gathering host information, 185
measuring bandwidth utilization, 174
measuring disk I/O, 173
measuring memory-related activity, 180–181
measuring processor utilization, 22, 170–171
measuring processor utilization by individual processes, 172
measuring queue lengths, 175
playtime in testing process in agile development and, 335
skills need by performance engineers and, 8
virtual machines mimicking, 316
Workloads, see also Load
applying numerical data to the characterization of, 99–103
identifying, 85–87
mapping application domains to, 91–95
overview, 85
performance requirements designed for size of, 107–108
pitfalls, 321
reference workloads in different environments, 87–88
specifying numerically, 95–99
summary and exercises, 103–104
time-varying demand and, 88–91
This page intentionally left blank