# SYSTEM PERFORMANCE TESTING

## A CASE STUDY

*Part 1: Developing the Test Strategy*

**Ross Collard**
Collard & Company
New York, NY

Version 5.5, February 2005

# *Table of Contents – Part 1 of the Case Study*


**INTRODUCTION TO THE CASE STUDY ..... 12**

OVERVIEW ..... 12
- The Rationale for this Case Study ..... 12
- Skills Needed for the Case Study ..... 13
- Organization of this Case Study ..... 13
- Learning Objectives ..... 14
- Disclaimers ..... 15
- The Assumptions behind this Book ..... 16
- Terminology ..... 19
- The Use of Mathematics in this Case Study ..... 19

USING THE CASE STUDY ..... 20
- The Concise versus the Full Version of the Case Study ..... 20
- A Friendly Warning ..... 20
- The Exercises ..... 21
- The Suggested Answers ..... 21
- Sequence of the Steps in the Test Methodology (and thus the Exercises) ..... 22
- Exercise Options ..... 22
- Time Required for the Exercises ..... 22

**I. THE CONCISE VERSION OF THE CASE STUDY ..... 23**

EXERCISE 1.1: REVIEWING THE PERFORMANCE TESTING OBJECTIVES ..... 23
- Objectives ..... 23
- Instructions ..... 23
- Questions to Address ..... 23

DESCRIPTION OF THE SITUATION, SECTION A: THE BUSINESS CONTEXT ..... 23
- Background ..... 23
- Your Role ..... 25
- Business Goals ..... 25
- Success Criteria ..... 25
- Testing Objectives ..... 26

EXERCISE 1.2: MODELING THE ARCHITECTURE ..... 26
- Objectives ..... 26
- Instructions ..... 26

DESCRIPTION OF THE SITUATION, SECTION B: THE SYSTEM ARCHITECTURE ..... 27

EXERCISE 1.3: SPECIFYING THE PERFORMANCE REQUIREMENTS ..... 29
- Objectives ..... 29
- Instructions ..... 29

EXERCISE 1.4: PERFORMING THE INITIAL IMPACT ASSESSMENT ..... 31
- Objectives ..... 31
- Instructions ..... 31
- Levels of Demand – Peak Load ..... 35
- Assumptions ..... 36

*Introduction to LAN Models ..... 36*
- LAN Model for Low Traffic ..... 37
- LAN Model for Moderate to High Traffic ..... 39

*LAN Utilization Worksheet ..... 43*

*LAN Utilization Worksheet ..... 47*

EXERCISE 1.5: DECIDING WHERE TO OBSERVE AND WHAT TO MONITOR ..... 49
- Objectives ..... 49
- Instructions ..... 49

# *Table of Contents – Part 1 of the Case Study*


EXERCISE 1.6: SELECTING THE METHODS OF TESTING ..... 49
Objectives ..... 49
Instructions ..... 49
EXERCISE 1.7: DETERMINING THE TEST FOCUS AND COVERAGE ..... 51
Instructions ..... 51
Questions to Address – Load Demand Risks ..... 51
Questions to Address – Infrastructure and Design Risks ..... 52
EXERCISE 1.8: CALCULATING THE TEST WORK LOAD ..... 54
Objectives ..... 54
Instructions ..... 55
Questions to Address ..... 55
*A. Test Work Load Volumes* ..... *55*
*B. Test Execution Logistics* ..... *61*
DESCRIPTION OF THE SITUATION, SECTION C: VOLUMETRIC ASSUMPTIONS ..... 62
User Demand ..... 62
Session Statistics ..... 62
Peak Loads ..... 63
EXERCISE 1.9: BALANCING EXPLORATORY AND STRUCTURED TESTING ..... 63
Objectives ..... 63
EXERCISE 1.10 REVIEWING A DETAILED TEST SCENARIO ..... 64
Objectives ..... 64
Instructions ..... 64
DESCRIPTION OF THE SITUATION, SECTION D: PERFORMANCE TEST SCENARIO ..... 65
Description of the Test Scenario ..... 66
EXERCISE 1.11 DESIGNING THE TEST ENVIRONMENT ..... 67
Objectives ..... 67
Questions to Address ..... 67
EXERCISE 1.12: ESTIMATING THE NUMBER OF TEST CYCLES ..... 67
EXERCISE 1.13: REVIEWING THE PERFORMANCE TEST PLAN ..... 67
Performance Test Plan Review Checklist ..... 68

**II. THE FULL CASE STUDY: UNDERSTANDING THE SITUATION ..... 79**

EXERCISE 2.1: REVIEWING THE PROPOSED TESTING OBJECTIVES ..... 79
Introduction ..... 79
Instructions ..... 79
Questions to Address ..... 80
FOLLOW-UP: TEAM DISCUSSION OF THE TESTING OBJECTIVES ..... 82
Instructions ..... 82
DESCRIPTION OF THE SITUATION, SECTION 2.A: THE BUSINESS CONTEXT ..... 83
*A.1 Overview* ..... *83*
A.1.1 Context ..... 83
A.1.2 Objectives ..... 83
*A.2 The Business Background* ..... *83*
*A.3 Your Responsibilities* ..... *84*
*A.4 The Basic Functions of the System* ..... *84*
*A.5 The Interfacing Systems* ..... *85*
*A.6 The Business Operations and Processes* ..... *86*
*A.7 System Work Flows* ..... *87*

# *Table of Contents – Part 1 of the Case Study*


*A.8 The Business Objectives for the System* ..... *88*
*A.9 The Performance Goals* ..... *89*
*A.10 System Constraints* ..... *90*
A.10.1 Usability ..... 90
A.10.2 Data Availability and Integrity ..... 90
A.10.3 Security ..... 91
A.10.4 Maintainability ..... 91
*A.11 Assessment of the Current System Performance and Robustness* ..... *92*
*A.12 The Performance Testing Objectives* ..... *92*
*A.13 Trade-offs* ..... *93*
EXERCISE 2.2: MODELING THE ARCHITECTURE ..... 94
Introduction ..... 94
Instructions ..... 94
DESCRIPTION OF THE SITUATION, SECTION 2.B: THE SYSTEM ARCHITECTURE ..... 98
*B.1 Architecture Overview* ..... *98*
B.1.1 Infrastructure Design Goals and Principles ..... 100
B.1.2 Logical Vs. Physical Design ..... 101
B.1.3 Design Review and Validation ..... 103
*B.2 Designing for High Availability* ..... *103*
B.2.1 Designed-In Redundancy ..... 103
B.2.2 Designed-In Scalability ..... 104
B.2.3 Clustering and Fail-Over ..... 104
B.2.4 Geographic Dispersion ..... 104
B.2.5 Locations and Assignments of the Servers ..... 105
*B.3 Major Tiers and Work Load Distribution* ..... *105*
B.3.1 The Front-End ..... 106
B.3.2 The Back-End ..... 106
B.3.4 Load Balancing ..... 107
B.3.4.1 Network Load Balancing for the Front-End ..... 107
*B.4 The Web Sites* ..... *108*
B.4.1 The Primary Web Site ..... 108
B.4.2 Providing Web Services ..... 108
B.4.3 Proxy Servers ..... 109
B.4.4 Web Databases ..... 109
B.4.5 Location of Web Content Storage ..... 109
B.4.6 The Secondary Web Site ..... 110
*B.5 The Data Architecture* ..... *110*
B.5.1 The Data Content ..... 110
B.5.2 Data Conversion ..... 110
B.5.3 Database Size ..... 111
B.5.4 The Database Servers ..... 111
B.5.5 Data Distribution and Mirroring ..... 111
*B.6 Networks and Communications* ..... *112*
B.6.1 The Network Topology at Headquarters ..... 112
B.6.2 Network Interface Cards (NICs) ..... 113
B.6.3 Utilization of Network Technologies ..... 113
B.6.4 The E-Mail and Fax Servers ..... 113
B.6.5 The Voice Telephone Servers ..... 114
B.6.6 The Wireless Routers or Servers ..... 114
B.6.6 The West Coast Remote Location Servers ..... 114
*B.7 Other Subsystem and Component Descriptions* ..... *115*

# *Table of Contents – Part 1 of the Case Study*


B.7.1 The Application Servers ... 115
B.7.2 The Print Servers ... 115
B.7.2.1 Print Out-Sourcing ... 115
B.7.3 The Support Software ... 115
*B.8 Security Considerations ... 116*
B.8.1 IP Addresses ... 116
B.8.2 Firewalls ... 116
B.8.3 Trade-offs of Security and Performance ... 116
*B.9 Scalability Considerations ... 117*
B.9.1 Application Processing Scalability ... 117
B.9.2 Database Scalability ... 117
B.9.3 Network Scalability ... 117
*B.10 System Implementation ... 117*
B.10.1 Re-Use of the Existing Equipment ... 117
B.10.2 The System Implementation Strategy ... 118
B.10.3 Physical Installation and Set-Up of the Equipment ... 118
*B.11 Architecture Evaluation ... 118*
B.11.1 Review History ... 118
B.11.2 Likely Performance Vulnerabilities ... 119
B.11.3 Possible Bottlenecks ... 119
B.11.4 Test Suggestions from the Technical Community ... 120
*B.10.4.1 Database Performance ... 121*
*B.10.4.2 Web Site Performance ... 121*
*B.10.4.3 Maintainability ... 121*
EXERCISE 2.3: SPECIFYING THE PERFORMANCE REQUIREMENTS ... 122
Instructions ... 122
EXERCISE 2.4: PERFORMING THE INITIAL IMPACT ASSESSMENT ... 124
Instructions ... 125
Simplifying Assumptions ... 127

**III. DETERMINING THE PERFORMANCE TEST APPROACH ... 131**

Introduction ... 131
EXERCISE 2.5: DECIDING WHETHER TO OUTSOURCE ... 131
Instructions ... 131
Introduction ... 132
*Advantages of Outsourcing ... 133*
*Disadvantages of Outsourcing ... 134*
*Outsourcing Work Sheets ... 135*
Advantages of Outsourcing ... 135
Disadvantages of Outsourcing ... 136
EXERCISE 2.6: SELECTING THE METHODS OF TESTING ... 137
Instructions ... 137
Test Methods Work Sheet ... 138
EXERCISE 2.7: DETERMINING THE TEST FOCUS AND COVERAGE ... 140
Instructions ... 140
*Questions to Address (A: Primary Factors) ... 141*
*Questions to Address (B: Secondary Factors) ... 141*
EXERCISE 2.8: CALCULATING THE TEST WORK LOAD ... 144
Instructions ... 144

# *Table of Contents – Part 1 of the Case Study*


Questions ..... 144
*A. Test Work Load Volumes* ..... *144*
*B. Test Execution Logistics* ..... *146*
DESCRIPTION OF THE SITUATION, SECTION 2.C: VOLUMETRIC ASSUMPTIONS ..... 147
*C.1 Measurements and Assumptions* ..... *147*
*C.1.1 User Demand* ..... *147*
*C.1.2 Session Statistics* ..... *147*
*C.1.3 Hits and Views* ..... *148*
*C.1.4 Peak Loads* ..... *148*
EXERCISE 2.9: BALANCING EXPLORATORY AND STRUCTURED TESTING ..... 149
Instructions ..... 149
*How Much Do We Know?* ..... *151*
EXERCISE 2.10: DEVELOPING YOUR TEST AUTOMATION FRAMEWORK ..... 151
*Assessing Readiness for Test Automation* ..... *151*
*Designing the Automation Framework* ..... *152*
EXERCISE 2.11: ESTIMATING THE NUMBER OF TEST CYCLES ..... 153
EXERCISE 2.12: DEFINING THE ROLES AND RESPONSIBILITIES ..... 154
EXERCISE 2.12 BUILDING FLEXIBILITY INTO THE PERFORMANCE TESTING ..... 154
EXERCISE 2.13 COORDINATING PERFORMANCE TESTING WITH OTHER ACTIVITIES ..... 155

**IV. SPECIFYING THE TESTS ..... 156**

EXERCISE 2.14: DEVELOPING THE PERFORMANCE TEST SCENARIOS ..... 156
Instructions ..... 156
DESCRIPTION OF THE SITUATION, SECTION 2.D: PERFORMANCE TEST SCENARIOS ..... 158
*D.1 A (Claimed) High-Opportunity Test Scenario* ..... *158*
*D.2 A Detailed Version of the Test Scenario* ..... *158*
D.2.1. Description ..... 158
D.2.2. Purpose and Intended Use ..... 159
D.2.3. Justification ..... 161
*D.2.3.1 Possible Outcomes* ..... 161
D.2.4. Target Audience ..... 161
D.2.5. Performance Requirements addressed by this Scenario ..... 162
D.2.6. Hypotheses to be Proven or Disproven ..... 162
D.2.7. Description of the Test Scenario ..... 163
D.2.8. Test Infrastructure ..... 163
*D.2.8.1 Test equipment* ..... 163
6.2.9. Pre-conditions ..... 165
6.2.10. Expected Actions ..... 165
6.2.11. Timings ..... 165
6.2.12. Post-conditions ..... 166
6.2.13. Test Work Loads ..... 166
6.2.14. Types of Testing to be Utilized ..... 167
6.2.15. Types of Testing that are NOT included in this Project ..... 167
6.2.16. Automated Test Scripts Used in this Scenario ..... 167
6.2.17. Manual Test Scripts ..... 169
6.2.18. Data Collection Plans ..... 169
6.2.19. Data Interpretation and Analysis Plans ..... 169
6.2.20. Results Evaluation and Reporting ..... 169

# *Table of Contents – Part 1 of the Case Study*


6.2.21. Proof of Concept / Trial Run ..... 170
6.2.22. Project Management and Status Tracking ..... 170
EXERCISE 2.14: SELECTING THE TEST TOOLS ..... 170
Instructions xxx ..... 170
EXERCISE 2.15: USING THE TEST TOOLS ..... 170
Instructions ..... 170
Questions to Address ..... 171
EXERCISE 2.16: COLLECTING THE PERFORMANCE DATA ..... 171
Questions to Address ..... 171
EXERCISE 2.17: ANALYZING THE PERFORMANCE DATA ..... 171
Questions to Address ..... 171
EXERCISE 2.18: IDENTIFYING AND REVIEWING THE OUTSTANDING ISSUES ..... 172
Questions to Address ..... 172
DESCRIPTION OF THE SITUATION, SECTION 2.E: SUPPORTING INFORMATION ..... 172
*E.1 System Usage Demographics ..... 172*
*E.1.1 Timing of the Occurrences of Peaks ..... 173*
*E.2 Feature List and Operational Profile ..... 173*
*E.2.1 Customer Service Group ..... 173*
*E.2.2 Catalog Publishing Group ..... 174*
*E.2.3 Warehouse Group ..... 175*
*E.2.4 Information Systems Group ..... 176*
*E.2.5 Senior Management Group ..... 176*
*E.3 Transaction Lengths ..... 176*
E.3.1 Customer Service Group ..... 177
E.3.2 Catalog Publishing Group ..... 177
E.3.3 Warehouse Group ..... 177
E.3.4 Information Systems Group ..... 178
E.3.5 Senior Management Group ..... 178
*E.4 Other Systems on the Shared Infrastructure ..... 178*
*E.4.1 Frequency of Utilization ..... 178*
E.4.1.1 Billing Group ..... 179
E.4.1.2 Publisher Ordering Group ..... 179
E.4.1.3 Marketing Group ..... 179
E.4.2 Transaction Lengths ..... 179
*E.5 Growth Projections ..... 179*
*E.6 Changing Mix of Demands ..... 180*
*E.7 Service Level Agreements (SLAs) ..... 180*
*E.8 System Development and Feature Testing Methodologies ..... 181*
*E.9 Automated Test Facilities ..... 182*
*E.10 Test Conditions and Constraints ..... 182*
EXERCISE 2.19: TRANSITIONING TO POST-DELIVERY LIVE PERFORMANCE MONITORING ..... 183
Introduction ..... 183
EXERCISE 2.20: TEAM DISCUSSION OF THE REMAINING TEST ISSUES ..... 183
Instructions ..... 183

# *Table of Contents – Part 1 of the Case Study*

**APPENDICES....184**

*Appendix A. Basic Definitions and Concepts....184*
*Appendix B. Establishing Performance Requirements....239*
An Example of a System Performance Requirement....239
The Process for Setting Requirements....240
Setting Performance Requirements Early....241
Mapping from the User's Perspective to the System Administrator's....242
Including a Performance Focus in the System Design....242
Defining the Workloads....243
Bypassing Load Calculations....243
Performance Objectives....243
*Appendix C. The Initial Impact Assessment....244*
Introduction....244
The Purpose and Nature of the IIA....244
Types of Impact Assessment....245
The Scope of the IIA....245
Prioritizing the Performance Test Needs....246
What Situations Need to be Assessed?....248
Prioritizing Within Systems....248
The Prioritization Process....249
When to Conduct the IIA....250
Potential Funding Issues....250
*Appendix D. Roles and Responsibilities....251*
Overview....251
The Role of the CIO and Senior Managers....251
The Role of System & Network Administrators....252
The Role of the Testers....253
The Performance Test Team....253
A Caution....253
*Appendix E. Performance and Robustness Testing Methods....254*
Approaches to Testing....254
1.0 Testing which is driven by what we want to measure....254
2.0 Testing which is based on the source or type of the load....254
3.0 Testing which seeks to stress the system or find its limits....255
4.0 Testing which focuses on the impact of changes....255
*Appendix F: Challenges in Performance and Robustness Testing....256*
Why is Performance Testing Difficult?....256
Common Issues of the Testers....256
A. Background Knowledge....257
*A1. Understanding of the System and its Context....257*
*A2. Slope of the Learning Curve....257*
*A3. Availability of a System Model....257*
*A4. The System Scope and Boundaries....257*
B. The Testware....258
*B1. Adequacy of the Test Facilities....258*
*B2. Adequacy of the Test Work Loads....258*
*B3. The Overhead and Logistics of Testing the System....258*
*B4. Unanticipated Glitches in the Test Environment....259*
C. The Live and Test Environments....259
*C1. Environment-Specific Issues....259*
*C2. Mixed Environments....260*

# *Table of Contents – Part 1 of the Case Study*


*C3. The Impact of New and Improved Technologies* ..... 260
*C4. Large Numbers of Testable Configurations* ..... 260
D. The Testing Tools ..... 261
*D.1 Expense and Complexity of the Tools* ..... 261
*D.2 Tool Limitations* ..... 261
*D.3 Mis-Use of Tools* ..... 262
E. The Test Methods ..... 262
*E.1 Testability* ..... 262
*E.2 Deciding What to Measure* ..... 262
*E.3 Lack of Repeatability* ..... 262
*E.4 Interpretation of Results* ..... 263
*E.5 Problem Isolation* ..... 263
*E.6 Validity and Credibility of the Results* ..... 263
F. Project Management ..... 263
*F.1 Late Occurrence in Projects* ..... 263
*F.2 Testing without Pre-defined Exit Criteria* ..... 264
*F.3 "Trivial" Changes which have Widespread Consequences* ..... 264
*F.4 Specialized Expertise Required* ..... 264
*F.5 Coordination of Specialists* ..... 264
*F.6 Lack of Candor* ..... 265
*F.7 Unexamined Information* ..... 265
*F.9 Lack of Buy-In* ..... 265
*Appendix G. The Test Automation Framework* ..... *266*
Integration of the Test Tools ..... 275
*Appendix H. Test Automation Activities* ..... *277*
A. The Overall Approach ..... 277
B. Support ..... 277
C. Review of the Current Situation ..... 277
D. Justification of Test Automation ..... 278
E. Assessment of Automation Effectiveness ..... 278
F. Outsourcing ..... 278
G. Automation Start-Up ..... 278
H. Planning for Test Automation ..... 278
I. Automated Functional and Performance Test Plans ..... 278
J. People and Organization Factors ..... 279
K. Tool Utilization ..... 279
L. Test Policies and Procedures ..... 280
M. Automation of System Requirements and Design ..... 280
N. Automation of System Development and Maintenance ..... 280
O. Performance Testing ..... 280
P. Robustness Testing ..... 281
Q. Test Environment Management ..... 281
R. Test Data Management ..... 281
S. Test Case Development ..... 282
T. Test Execution ..... 282
U. Problem Management and Resolution ..... 282
V. Manual versus Automated Testing ..... 282
X. Test Project Management ..... 282
*Appendix I. Avoiding Performance Surprises* ..... *283*
1. Early Design Reviews ..... 283
2. Use of Prototypes for Load Testing ..... 284
3. Performance Prediction ..... 284

# *Table of Contents – Part 1 of the Case Study*


4. Early Check-out of the Test Facilities ..... 284
5. Early Component-Level Performance Testing ..... 284
6. Early Trial Full-System Load Testing ..... 285
7. Use of Simulators for the Test Environment ..... 286
8. Designing for Performance ..... 286
10. The Performance Review ..... 288

*Have Timely Access to the Reviews* ..... 289
*Review the System Architecture* ..... 289
*Understand How the System Works* ..... 289
*Review the Process Flow Model* ..... 289
*Understand How the System will be Used* ..... 289
*Walkthrough the Topology to Look for Bottlenecks* ..... 289
*Review the Operational Track Record of similar other Systems* ..... 290
*Review the Use of Shared Resources* ..... 290
*Determine How to Minimize the Wait Times* ..... 290
*Decide Where to Focus the More In-Depth Performance Review* ..... 290
*Review each High-Potential Component* ..... 291
*Review the Efficiency of Computations* ..... 291

11. Designed-In Flexibility for Tuning ..... 292
12. Review Questions ..... 292
13. Designing for Testability: Assessing Testability ..... 294
14. The Characteristics of a Testable System ..... 295

*Appendix J. System Architecture Diagrams* ..... *296*
J.1 User-Oriented Architecture ..... 296
J.2 Function-Oriented Architecture ..... 297
J.3 Geographically-Oriented Architecture ..... 298
J.4 Device-Oriented Architecture ..... 299

*Appendix K. Template of the Performance Testing Work Plan* ..... *300*
Part A: Test Preparation ..... 300
Part B: Execute the Tests and Evaluate the Results ..... 305

## About Ross Collard

The author of this book, Ross Collard, is president of Collard & Company, a consulting firm which is headquartered in New York City and specializes in software testing and quality. His clients have included ADP, Alcatel, American Express, Anheuser-Busch, Apple, AT&T, Bank of America, Blue Cross/Blue Shield, Cisco, Computer Associates, Dayton Hudson, Dell, EDS, Exxon, General Electric, Goldman Sachs, Hewlett-Packard, Hughes Aircraft, IBM, Intel, JP Morgan, Merck, Microsoft, Motorola, Nortel, Novartis, Novell, Procter & Gamble, Prudential, SIAC, Siemens, State Farm Insurance and Verizon. His government clients have included NASA, Federal Reserve Bank, State of California and U.S. Air Force.

Ross has conducted seminars on software testing and quality for businesses, governments and universities, including George Washington, Harvard and New York Universities and U.C. Berkeley. He has a BE in Electrical Engineering from the University of Auckland, New Zealand, an MS in Computer Science from the California Institute of Technology and an MBA from Stanford University's Graduate School of Business. He can be reached at rcollard@attglobal.net or 1 (212) 941-5962.

## Acknowledgements

I would like to gratefully acknowledge the help of the many people who have reviewed, commented and contributed to this book. They include James Bach, Scott Barber, Michael Bolton, Julian Harty, Linda Hamm, Doug Hoffman, Paul Holland, Dave Jewell, Chris Johnson, Philip Joung, Cem Kaner, Mike Kelly, Nancy Landau, Brian Marick, Jude McQuaid, Daniel Navarro, Hung Nguyen, Noel Nyman, Dale Perry, Avinash Persaud, Bret Pettichord, Martin Pol, Greg Pope, Johanna Rothman, Rob Sabourin, Drew Slikowsky, Steve Splaine, Roland Stens, Elaine Weyuker and Karl Wiegers.

A good deal of credit for this book has to go to my consulting clients over the years, for providing both real-world challenges to learn from and the income to sustain the writing of these books. I would also like to thank the more than 50,000 students who have attended our seminars on software testing and quality assurance from the period from 1990 to the present, and who have contributed many insights, found many bugs in the materials, and have asked many penetrating questions which have stretched my thinking.

# Introduction to the Case Study

This case study describes a situation in which a performance test is needed. Your assignment is to identify and analyze the business and technical issues and develop a test strategy for the situation. This strategy describes how you will use testing to predict the system's actual performance, in order to assess whether it will be acceptable in live operation.

## *Overview*

### The Rationale for this Case Study

Today's personal computers and even mobile phones run hundreds of thousands of times faster than IBM's first computer. Network speeds have increased and database storage costs have dropped by a factor of more than a million. System performance engineering and software optimization have advanced from sparse, hit-or-miss rules of thumb to quasi-respectable professional fields. So why bother to test system performance and robustness? I'd like to describe the problem by telling a story. Imagine that you have made a heroic effort to deliver a system under unreasonable deadlines and with limited staff and equipment. You scrupulously test to ensure the features work as expected, and then release the system. You monitored the performance as a byproduct of feature testing, and everything seemed fine in the test lab. A few days later, you receive a call from a senior executive in the client community. You are expecting words of appreciation, but he can only moan and rant about what you have done to him. Our story ends on a sad note. The system's features work, but:

- The system response time is slow.
- The throughput (the volume of concurrent demands that the system can handle), is low.
- The system cannot handle peak loads or sudden surges in demand.
- It is fragile and readily crashes or hangs, and cannot recover from errors.
- It does not work the same on all the users' workstation configurations.

- Intermittently, user actions that should be independent and isolated other apparently interact and interfere with each other. These problems occur under heavy load but cannot be observed under normal load.

- The system works fine in the test lab but does not scale up – there appear to be hidden bottlenecks.

- The system's resource use is prohibitively expensive. It is a "resource hog".

Ouch. If you identify with this story -- you have "been there, done that" -- and would prefer not to re-live the experience, then this case study is for you. Or if you have never been there and want to keep it that way, this will help you too.

**Skills Needed for the Case Study**

This case study and its exercises do not require specialized expertise in any particular field, such as e-commerce system design, queuing theory, statistical sampling or network engineering. They do require a familiarity with information systems technology, such as a sense of what a local area network (LAN) does or how a database works.

**Organization of this Case Study**

The case study contains six main parts:

*Part 1, Developing the Performance Test Strategy:* In this part, you will address how to test a system's performance. The strategy you develop will include the test objectives and focus, justification, scope, technical and business issues, risk factors and your recommendations for the overall test approach.

*Part 2, Developing the Test Strategy -- Suggested Answers to Part 1:* You will critique suggested answers to the questions in Part 1, and either concur or suggest revisions and improvements to these answers. Part 2 is not complete -- answers have not been published yet for all the exercises in Part 1.

*Part 3, Reviewing the Proposed Test Strategy:* You will review and issue an opinion on a proposed test strategy. This strategy is presented in the form of an executive summary and a comprehensive set of appendices with the supporting details. (Parts 3 through 6 are not included in this set of documents.)

## *Introduction to the Case Study*

*Part 4, Reviewing the Consultants' Report:* Subject matter experts (the consultants) have rendered an opinion on the adequacy of the proposed test strategy in Part 3. Your job is to determine which of the consultants' findings and conclusions are valid and worth acting on, and what corrective actions to take.

*Part 5, Reviewing a "Practical" Test Strategy:* In this part, an alternative approach is presented, which was developed based on the consultants' feedback. You will review and compare this alternative to the original proposed approach, and determine which of the two strategies is the most suitable, or what mix of the two you recommend.

*Part 6, Developing the Robustness Test Strategy:* You will review and critique the proposed strategy for testing the systems' reliability and recoverability.

**Learning Objectives**

The lessons you will learn from this case study and series of exercises include:

- Planning and preparing for effective performance testing:
- How to develop a performance test strategy in a typical mixed-technology, mixed-vendor environment with multiple interdependent application systems.
- How to set performance goals and testing objectives.
- How to perform a risk assessment and use it to focus and prioritize the test efforts.
- How to design the test lab, and allow for the differences between the lab and the real-world infrastructure which will be utilized in live operation.
- How to design realistic test work loads.
- Executing performance tests and evaluating results:
- How to use automated load testing tools effectively, and minimize opportunities for tool mis-utilization.
- How to decide what to measure and what data to collect.
- How to organize and run a performance test.

- How to interpret the harvest of performance data and form meaningful, trustworthy conclusions about performance in live operation.

- Why and how system performance testing is fundamentally different from performance engineering, system optimization and feature testing.

**Disclaimers**

The organizations described in this case study are fictional, and not intended to represent any real organizations.

Testing tools from various vendors are mentioned in this case study and its accompanying series of exercises. None of these mentions should be construed as an endorsement of a particular vendor or a recommendation of a particular tool.

Trademarked names, such as the names of software testing tools, appear in this book. I am using these names for illustration purposes only, with no intention of infringing on the trademarks. Not all the tool information in this book will be up to date. There are many testing tools on the market, and the vendors continually introduce new ones and frequently upgrade the features of existing ones. For the latest and most accurate information, check with the vendors directly.

The book draws on a great many ideas from many sources. I provided credit in the section entitled “Acknowledgments”, but there still may be some use of published or unpublished ideas which I have used without proper attribution to the original author(s). This misuse is inadvertent, and I will correct it as promptly as possible when I am notified.

Since the primary target audience for this book is working practitioners rather than academic researchers, the book does not contain extensive footnotes, academic citations and an exhaustive bibliography of reference works. The ideas in this book have come mainly from my and others’ on-the-job experiences, not from a literature search.

The book contains quantitative information to help describe software testing norms, poor and best practices. Much of this information is not based on scientifically rigorous, statistically valid experiments. Software and system quality are fast moving fields where there is a dearth of rigorous data. Despite the lack of rigor in its collection, information which appears reasonably accurate based on my experience is included to provide as full a picture as possible.

Microsoft has published a similar performance testing case study, for a fictional firm called the Duwamish Book Store. The web-based business functions supported by the system in their case study include point-of-sale, order entry, shipping and receiving, and a book catalog. The case study in this book was published in a substantially complete version well before Microsoft published theirs. The Duwamish Book Store case study is worth reviewing as a counterpoint to this one. Microsoft's case study is simpler and less rich in issues, so it may be worth looking at first, before you get into the complexities and nuances of this case study.

All errors are the responsibility of the author, who would appreciate any feedback about defects or suggested improvement opportunities.

**The Assumptions behind this Book**

I have written this book based on the following assumptions.

You, the reader of this book, are intelligent and curious. You are motivated to learn how to become more effective as a tester, believe that quality is important and that you have an important role to play in software quality. You enjoy the satisfaction of a job well done.

You are willing to work in order to gain an important benefit. Most exercises in this book take at least 30 to 45 minutes and many take hours to answer well, even though the most time-consuming part has already been done for you, which is the gathering and organization of the data needed to answer the questions. If you zip through an exercise in 5 to 10 minutes, you may feel like congratulating yourself for being so smart. But it is unlikely that you have gotten the full value from the exercise and seen its important nuances.

You do not work in an overly hostile environment. You can ask questions on the job without the risk of looking stupid, and the other busy people who you work with are cooperative. You have access to people like software engineers and system users to get your questions answered.

You will not have to master any specialized terminology in order to read this book. Unfortunately, there is no universal consistency of terminology in the testing world. For example, people use terms like test case, test script, test condition and test scenario interchangeably, or alternatively can mean different things to different people. In an effort to be rigorous, I could have begun this book with a long series of definitions. This specialized terminology, while precise, would probably be outside the common vernacular of most testers, so you the reader would have to mentally translate as you read. In addition, I did not want you to have to wade through many definitions before

getting to the substance. I can promise, however, that the ways in which I use terms in this book are as much or more in the mainstream of common test usage than any other terminology. Appendix A provides a glossary of terms.

You are always working under deadline pressures, and you may not always have the luxury of crossing every "t" according to some formal testing methodology. Nevertheless, the deadlines which "they" impose on you are merely ambitious, feasible but not impossible.

Being able to analyze, reason, observe, evaluate and think critically is much more important to your success as a tester than being able to mindlessly follow directions, or mindlessly filling out forms so that your test case documentation looks great in a superficial review.

You do not always have complete, correct and well-written functional specifications for the systems you are testing. Unfortunately, specs like those are a fantasy for most testers. If we had those great specs and perfect information, the testers' job would be easy. The hard part is getting usable, specific and trustworthy information in the first place, and analyzing the specs for testability.

The real world is a lot more complicated and messy than the worlds assumed in many books. We cannot ignore real-world complications like the volatility of testing situations in this fast moving world. Dealing with rapid and last-minute changes in the system functionality and technical environment, and bug fixes which introduce inadvertent side effects, etc., is a way of life for most testers.

In this book, we grapple with the real-world issues that testers face every day, like handling deadline pressures, vague functional specs., managers who do not understand testing, uncooperative software engineers, staff turnover, gremlins in the test environment, etc. These complications are not so overwhelming that it becomes impossible for you to do your job, though. In that situation, you don't need a book, you need a fire extinguisher.

You do not know how to develop a test plan or need to read a book on test planning as a prerequisite to this book. Test case design (micro-planning) is done within the context and framework of a test plan (macro-planning), which established the overall objectives, scope and approach for a test project. In the exercises, we will assume that the test team leader has already prepared a test plan or otherwise provided the direction you need for developing the test cases.

## *Table of Contents – Part 1 of the Case Study*

All testing is context-specific. The exercises in this book add value, by showing examples of how the techniques work, but you will still have to examine for yourself how each technique applies in your situation. I suggest that as you read each section of the book you continually ask yourself:

- Will this technique be helpful for me?

- If so, how, when and where can I apply it in my work?
- What support do I need from my boss or others to make this succeed for me?

Finally, I suggest that you do not view this book as eternal truth, but view it with a critical eye and apply common sense to determine whether and how its recommendation will work for you.

Welcome to this book, and may you have a great learning experience.

**Terminology**

There is no universal consistency of terminology in the field of testing. My choice has been to use the popular vernacular. I can promise that the terms used in this book are as much or more in the mainstream as any other. This means that the terms I use are not necessarily consistent with standards like the IEEE glossary or ISO. In my observation, the definitions of terms in these standards are little used by working practitioners. Put in another way, we'll use the equivalent of everyday Italian, not high church Latin. Appendix A provides a glossary of terms.

**The Use of Mathematics in this Case Study**

This book contains a little math, the minimum that we can get away with. Specialists like statisticians have evolved specific notations for mathematical expressions, which are the accepted standards within the communities of specialists. Since this is not a book for specialists, though, I have not attempted to comply with those notations. I have not published any mathematical proofs and have not strived for mathematical rigor.

This book is pragmatic and based on my and others' experiences, observations and rules of thumb. It may be dismissed by a theoretician as voodoo and superstition. The problem is that many of the topics of interest lie in relatively unknown territory – we simply do not have the theories, models and equations with which to compute the answers to our questions. In my opinion, there are rich and fascinating opportunities for applying mathematical and statistical techniques to performance testing. These techniques include design of experiments, sampling, factor analysis and ANOVA. However, they are outside the scope of this book.

## *Using the Case Study*

### The Concise versus the Full Version of the Case Study

Case studies usually are not scalable, which leads to a dilemma. Either we can use a "dumbed down" version of a case study with exercises that are fairly easy, but is too simplistic to see how its lessons apply to the real world. Or we can use a case study which faithfully reflects the full harshness of reality, but contains so many complexities, nuances and uncertainties that nobody can do the exercises.

To resolve the dilemma, this book contains two parallel case studies. One is for a small business with a web site and a few workstations on a local area network. The other is for a much larger variation of the same business, with hundreds of times more revenue than the small business, and with hundreds of times more system traffic. The case studies are parallel in that you will address the same series of questions to develop a test strategy, but your answers will be different based on the context. The concise version of the case study contains quicker and easier exercises. The full version contains complications which are non-existent or transparent in the concise version, so comparing the two provides a sense of the scalability of the issues and solutions. You can work through the concise version, the full version, or both.

### A Friendly Warning

The questions in the exercises may seem easy at first reading, but they can be difficult to answer. They require us to use judgment and think the situation through. They are also hard because the information that you are provided (as you will see in the next few pages) is not perfect -- just like in the real world. There is incomplete information, some uncertainty about the credibility and accuracy of portions of the information, unresolved issues and information which is open to interpretation in this case study. Sometimes testers get into "analysis paralysis" -- they feel that they cannot do anything at all unless they first know everything about the situation. However, we cannot wait until we have perfect information; the test strategy will never be formed.

Ironically, there also is information overload – whereas in some areas we are not told enough information, in others we may feel overwhelmed by details. Despite the temptation we sometimes have to want to know everything about a situation, attempting to master all the information available can be a daunting task. A great deal of information is presented in the next few pages, as background for the exercises, and an important part of the test planners' job is simply to become comfortable with the situation. I am not talking about ignorance-is-bliss comfort, but the confidence that comes from asking probing questions and realizing that the answers make sense to us

– we have a coherent model of the situation in our heads. The testers have to collect and review the information, determine what's relevant and sift out what's not, and decide what to question and where to probe further. Just like in a real test project.

**The Exercises**

To get the full value from this book, do the exercises. Because the questions you will be asked to address are not that easy, there is a temptation to disregard them both here and on a real-life project and "forge ahead" regardless. Full speed ahead -- damn the torpedoes! A great idea – until we hit a torpedo. These case studies come with their share of torpedoes. Though we do not want to become paralyzed in indecision, it is not a good idea to skip the questions in the exercises. If we do not think them through at the beginning of a testing project, sooner or later we will be forced to return and reconsider them, and then perhaps will have to painfully change the direction of the project.

I have designed this case study with the intent that you will work through the series of exercises in sequence, not skip them or only read their suggested answers. Nevertheless, each exercise is designed to stand alone if it is not possible for you to first complete the preceding exercises. If you do bypass the predecessor exercises, I recommend that you review their suggested answers before continuing. (The suggested answers are in not in this document but are Part 2 of the case study.)

You can do the exercises individually or with a small team, where you compare your thoughts with peers'. Some exercises involve group discussion and thus are not suitable for individuals working alone. If you are working through the case study by yourself, simply skip these team exercises.

**The Suggested Answers**

A powerful way to learn is to review and critique a suggested answer after you have developed, or tried to develop, your own. We call them suggested answers but not model answers or official answers, because often there is no one answer that we can agree is the best or only answer.

The answers are physically separated from the exercises, to help manage the temptation to peek ahead. Peeking at the answer is OK if you become stuck on a question, but if overdone it short-circuits the concept of self-discovery, your learning-by-doing. My intention is that you will do each exercise in turn, as presented in this book (Part 1), then review and critique the answer in Part 2 before returning to Part 1 for the next exercise. As mentioned earlier, Part 2 does not contain a full set of answers to the

exercises.

### Sequence of the Steps in the Test Methodology (and thus the Exercises)

I have presented the exercises as a one-way sequence of steps, such as: 1. gather data about a problem; 2. analyze data; 3. formulate solution; 4. implement solution. The world is usually not that simple.

On the job, you probably will not follow these steps in the presented sequence as if they are carved in stone, especially when a problem is complex. Instead, adapt the methodology to your situation, iterate, add or omit steps, parallel some activities and loop back to re-visit earlier steps, as appropriate.

### Exercise Options

You have three series of exercises to choose from in this book, either (a) the exercises accompanying the concise case study, numbered Exercise 1.1 through 1.14 which are quicker and easier to do, (b) the exercises for the full version of the case study, numbered 2.1 through 2.17 which are richer but require more effort, and (c), as a compromise, an abbreviated set of the exercises for the full version, where you get to work with the more realistic and challenging case study, but primarily review the suggested answers rather than developing your own answers.

### Time Required for the Exercises

I have provided an estimate of the time for you to allow for each exercise. These estimates are guidelines: some people complete an exercise in half of the estimated time while others take double the time. This is not a race. People who zoom through the exercises may fail to see the nuances, and deliver answers which are not well thought through. Give the exercises enough time to adequately absorb their lessons.

The total time you will require to do all the Part 1 exercises for the concise version of the case study is approximately 8 to 12 hours, and for the full version is approximately 32 to 40 hours.

# I. THE CONCISE VERSION OF THE CASE STUDY

## *Exercise 1.1: Reviewing the Performance Testing Objectives*

### Objectives

Your purpose in this exercise is to understand the business context of a typical performance testing situation, and critique the testing objectives.

### Instructions

Read the background to the case study in the attached Description of the Situation, Section A. Based on this background, answer the questions below. Each answer need be no more than a few lines long. Note that we are not looking for polished and detailed answers at this time, just an initial sketch of your thoughts, ready for discussion with the others in the class.

### Questions to Address

(1.1.1) What is the justification for this performance testing project?

(1.1.2) Is a performance goal the same thing as a performance test objective? (This is similar to the question, in functional testing: Is a system requirement the same thing as a test case?)

(1.1.3) Are the stated performance goals (a) relevant and significant to the business, and (b) testable (i.e., measurable and objective)?

(1.1.4) Are the stated performance testing objectives appropriate, and sufficient to guide the testing project?

## *Description of the Situation, Section A: The Business Context*

### Background

The PO system we have been discussing is part of an ordering system for our firm, Testing Little Books (TLB), a book club which specializes in testing and quality assurance books. “Little books” is the firm’s name for its line of proprietary pocket

books, which provide helpful hints and checklists for various types of testing such as security controls testing, database integrity testing and real-time embedded device testing.

The business has grown from a part-time, spare-bedroom operation to a staff of twelve full-time people, and needs a new system to support its business. TLB is building a new system for processing orders and shipping books, which will include a new web site and an intranet connecting the staff members. The intranet will be implemented through a combination of wired and wireless local area networks (LANs).

**Main Features of the System**

The new system will handle the following activities: ordering books, answering queries on the status of orders, book searches, printing catalogs for mailing, and e-mail broadcasting of special offers and promotions. Customers can perform these activities themselves directly via the web site, or can telephone or send mail. Using client workstations connected through the intranet, the book club staff members process these phoned-in or mailed-in orders, requests for refunds, etc.

TLB also sells books from publishers other than itself, hence the need for the PO system as described earlier by the use cases. The new system has been designed to accommodate modifications to support the sales of non-TLB books, and the book catalog in the system database lists a full inventory of books on testing and related topics. For the foreseeable future, the transaction volume for ordering books *from* publishers is small enough to ignore in this performance test project.

TLB outsources the printing and shipping of its books, so the detailed transactions needed for these functions are not included in the new system. However, the new system will send shipping instructions to the outsourcers.

**Sharing of Resources**

Another application, the billing system, will share the same infrastructure with the new ordering system. It generates invoices for the books ordered and tracks the payment status. The event that triggers generating an invoice is a shipment of books, which in turn is triggered by one or more orders.

A third application that shares the infrastructure is e-mail.

**Occasional Activities**

Other occasional activities which can place significant demands on the system resources include:

- Printing of paper catalogs for mailing (though 80% of catalogs are distributed electronically).

- Data mining and ad hoc report generation for senior managers, e.g., analyses of buying trends.

- Transmission of video and audio clips via the web site. These include brief interviews with authors and gurus in the software testing field, plus small complimentary tutorials.

**Your Role**

Your job is to check the performance of this new system before it goes live, in order to avoid problems in live operation, and your immediate assignment is to plan for this test.

**Business Goals**

The overall TLB business goals are to grow revenues, increase market share, improve profitability, and increase customer satisfaction.

**Success Criteria**

Providing acceptable system performance, availability, robustness and data integrity are considered critical to the success of this business. Without superior performance, for example, the senior managers believe the business will become marginalized. They are not willing to tolerate surprises when the system goes live that may imperil customer satisfaction and goodwill.

**Performance Goals**

In support of the business goals and success criteria, the system should meet these performance goals:

a) Response times must be competitive or beat the competition.

b) The system must be able to process normal work loads, heavy loads and the occasional surge in demand.

c) The system must be scalable, so it can be upgraded as necessary to accommodate growth for the first three years without undue delay and expense. (Any system can be upgraded if cost is not a factor.)

**Testing Objectives**

The primary objective is to provide a reasonable confidence that the performance goals will be met in the live operation, or to indicate what needs to be fixed in order to meet these goals.

## *Exercise 1.2: Modeling the Architecture*

**Objectives**

The system architecture is the blueprint for the infrastructure used in the live operation. The purpose of this exercise is to gain an understanding of how the system works by mapping its architecture into a diagram. This understanding later can help us to determine the test loads to run, the access points where we observe the system's behavior, the metrics to collect during these observations, and the target values and acceptable thresholds for the metrics.

**Instructions**

(1.2.1) Can we define system performance goals independently of the technical environment, specifically (a) how the system is implemented and (b) what environment and infrastructure it uses?

- By analogy, consider a situation where we need to test the performance of a delivery service that specializes in jewelry and other small precious items.
- To assess its performance, do we need to know if this delivery service uses carrier pigeons or aircraft?
- For what measures of performance, if any, is the service's use of carrier pigeons vs. aircraft irrelevant?
- For what other measures of performance, if any, is the service's use of carrier pigeons vs. aircraft relevant?
- Hint: ranking the efficiency of competing services, based on the jet fuel

consumed per package delivered, will place the pigeon-based services at the top of the list. Does this finding make sense?

(1.2.2) Read the description of the system architecture (see the Description of the Situation, Section B).

(1.2.3) Diagram a one-page model of the system architecture. For inspiration, see the attached examples of system architectures. In your diagram, show the major devices used by the system (such as servers, clients and routers). These terms are defined in the glossary. Do not be too concerned with adhering to a standard set of notations and graphic conventions, e.g., using a special symbol to represent each server. A simple rectangular box will do, provided it is clearly labeled. Link the devices together in the diagram to reflect the ways they will be connected in the live operation.

## *Description of the Situation, Section B: The System Architecture*

### Overview of the Major Components

The new system runs on two servers with mutual hot back-up and load balancing capabilities. Each connects to a database, a wired local area network (LAN), a gateway that provides Internet access, and through a router to a bank of printers. One of the servers also connects to a wireless router which drives a wireless LAN (WLAN). As mentioned earlier, the same servers also host the web site, handle e-mail, run the billing system and run security software such as firewalls.

Each server contains 4GB of semiconductor memory, a 200 GB hard drive, and a processor with a rated speed of 2 GHz.

### The Database

The database contains data about books in inventory, customers, orders, shipments and bills. This database has files on approximately 5,000 customers. At any time, the database contains about 1,000 open order records and 25,000 book inventory records.

Each server has a full copy of the database. The copies are kept synchronized by concurrent dual updates to both databases, and by a continual flow of monitoring transactions by the database management software (DBMS) to cross-check for consistency.

A database back-up is performed once nightly. It is expected to be completed in 15 minutes, with back-up transactions for 5,000 customer, 1,000 open order and 25,000

book inventory records within that period. The back-up is scheduled at a time when other demands are expected to be light. The database back-up traffic does not traverse the LANs; the back-up device is directly wired so that it bypasses the LANs. However, the back-up device is not stand-alone; a server must run the back-up job. During database back-ups, response times should not increase by more than 33% of their average values when no back-up is being run.

**The Intranet and LANs**

Ten of the client workstations and both servers will be connected by a wired Ethernet LAN, rated at 10 Mbps. No LAN router or switch is needed for such a small network; the LAN processing is handled by network interface cards (NICs) inserted into slots in the backs of the servers. There is a NIC inserted into every device connected to the LANs.

The voice phone traffic also traverses the wired LAN, because it connects the external phone services (VoIP provided by public telecom carriers), via a voice switch to the internal staffers in what is grandly named the call center. Peripheral devices such as printers, scanners and fax machines also connect to this wired LAN.

The remaining five client workstations are located in an office that cannot be wired without incurring the wrath of the landlord. This LAN connects these clients to a server via a wireless router, and to a few more peripheral devices. (There is only one wireless router, because the functions supported are not critical and a redundant link has been deemed unnecessary.) The wireless Ethernet LAN has a theoretical capacity of 11 Mbps (as per the original IEEE 802.3 standard), but to simplify the calculations we will assume 10 Mbps.

**The Web Site**

The web site is connected via dual gateways to a local Internet service provider (ISP). Each gateway connects from one of the two servers to the ISP, using a T1 line with a rated capacity of 1.544 Mbps. (There are two T1 lines in all.)

**The Voice Phone Service**

The organization’s voice telephone service is separately wired with its own self-contained switch and router. The telephone system will access the new system’s database, to look up the customer profile matching the phone number on each incoming phone call.

**The Support Software**

The decision has been made to use Windows XP on both the servers and the clients, and the database management software used by the system will be SQL Server.

## *Exercise 1.3: Specifying the Performance Requirements*

### Objectives

The purpose of this exercise is to assess the testability of the existing performance goals for the new system, and if necessary develop testable performance requirements.

### Instructions

(1.3.1) Review this example of a performance specification or requirement:

Performance Requirement Name: *Database Back-Up*

| **Transaction or Event Type** (Represented in the Load Mix) | **Throughput Load Volumes** (Transactions per Minute) | **Delivery Time or Response Time Goals** |
|---|---|---|
| 1. Customer record back-up | 333 | 15 minutes or less to back up 5,000 customer records; for 90% or more of the back-ups |
| 2. Order record and book inventory record back-up | 175 | 15 minutes or less to back up 1,000 order records and 25,000 book inventory records; for 90% or more of the back-ups |
| 3. Request page of the catalog | 7 | 4 seconds during the database back-up (i.e., 33% more than the normal goal of 3 seconds) or less, from the time when the request is submitted to when the requested page is displayed; 90% of the time or more |
| 4. Conduct book search | 7 | 9.3 seconds during the database back-up (i.e., 33% more than 7 seconds) or less, from the time when the search query is submitted to when the response is displayed; 90% of the time or more |

| | | |
|---|---|---|
| 5. Place a new book order | 7 | 9.3 seconds or less, from the time when the order is submitted to when the order acknowledgement or the error response is received; 90% of the time or more |
| 6. Request home page of the book club | 7 | 4 seconds or less, as measured at the visitors' workstations; 90% of the time or more |
| 7. Other background noise (concurrent unrelated activity) | 20 | Response time goals that have been set for the billing and e-mail systems |
| **Other Related Requirements** | **Aspects to Measure** | **Acceptable Thresholds** |
| 8. Resource utilization | Main processors, memory and network links | Utilization cannot exceed 65%, reserve capacity is not to fall below 35%. |
| 9. Error rate thresholds | Transactions listed above | Error rates not to exceed 1% for each system feature or transaction type |
| 10. Availability | Transactions listed above | All major system features available 99% of the time or more, 24 x 365, and with error rates that do not exceed the error rate thresholds |

(*) Each test case is defined to contain one transaction as an input stimulus and the related set of outcomes for that stimulus. All attempts to execute a test case are counted. Transactions are counted as errors if they are not completed because the system is unavailable, and are included in the error rate computation.

(1.3.2) For what situations should we set performance goals? We cannot define a performance requirement for every possible use of the system, so we need to focus on the most significant or the most representative.

a. List three situations where you believe performance will be critical or at least significant.

b. Describe a typical scenario which could serve as the bellwether indicator of the overall system performance. (In the stock market, a bellwether is a stock whose performance is indicative of the overall market condition and direction.)

(1.3.3) Develop the performance requirement for the situation you believe is the most critical one.

## *Exercise 1.4: Performing the Initial Impact Assessment*

### Objectives

The purpose of this exercise is to perform an initial assessment (IIA) of the system's likely performance issues, and decide whether a formal performance project is justified for the new system. The IIA is intended to be fairly quick and straightforward, and provide not exact but "good enough for now" answers.

We will limit the scope of the IIA to examining the wired LAN only, to simplify this exercise. (For the rest of this exercise, the term LAN means the wired network but not the wireless LAN.)

Normally, on the job we'd first ascertain whether the LAN is the best place to begin analysis. We want to begin with the system aspect or component that is easier to analyze and also may likely be a bottleneck. We'd also probably look at more than one aspect on the job, i.e., we might analyze the web services and the database as well as the LAN.

You will need to do some computations in this exercise; you may need a pocket calculator but not a supercomputer. And you do not need a mathematician, statistician or network engineer.

### Instructions

(1.4.1) In order to perform the initial assessment (IIA) of the system, first:

a. Review the attached information:
    i. The LAN models and worksheets.
    ii. The simplifying assumptions. Feel free to use these assumptions or replace them with your own. In that case, write them down.
    iii. The work load volumes to use in this calculation. (See the section entitled: "Levels of Demand – Peak Load".)

b. Analyze what we have available to work with (the usefulness and reliability of this information), before jumping into calculating the utilization of the network. The next three questions (1.4.2 through 1.4.4) address these issues. Only after they

have been addressed will we get into the calculations.

(1.4.2) How trustworthy will be the results of the calculations, given some uncertainty about the quality of the data provided?

a. In most situations, little or no data can be known with perfect accuracy. We need to factor uncertainty into our calculations. The table entitled: “Levels of Demand – Peak Load” provides estimates of the confidence we have in the input data.

a. Establish the target accuracy that is desired for your calculations.
    - These targets generally are expressed in terms like: “The real data value lies within 15% of the measured or calculated data value”.
    - A little later, we will add probabilities and levels of confidence to these expressions, but for simplicity we will ignore these refinements for now.
    - Be aware that a 1% improvement in the accuracy and precision of the computed answers can sometimes double the analytical time and effort.

b. To meet the target accuracy, how clean (i.e., how accurate and precise) does the input data need to be? This input data drives the calculations, for example, here it includes the transaction sizes (lengths) and volumes (occurrences). (We will re-visit this question below.)

c. How can we tell if the target accuracy has been attained or not?

d. Review the levels of confidence of the data provided in the section entitled: “Levels of Demand – Peak Load”. The levels of confidence range from low to high and are defined as follows:
    - Very Lo: there is a 50% or better chance that the real data value does not lie within 30% of the measured or reported data value.
    - Lo: there is a 50% or better chance that the real data lies within 15% of the measured or reported data value.
    - Mid: there is a 90% or better chance that the real data value lies within 15% of the measured or reported data value.
    - Hi: there is a 90% or better chance that the real data value lies within 5% of the measured or reported data value.
    - Very Hi: there is a 95% or better chance that the real data value lies within 5% of the measured or reported data value.

e. Given the levels of confidence expressed in the input data (transaction volume and length data), what is the likelihood that the real total network utilization will exceed the calculated total by more than 15%? More than 30%?

(1.4.3) Is it more appropriate here to use the mean (average) size or length of the LAN transactions, and not the median or the mode?

a. The terms mean, mode and median are defined here and in more depth in the glossary.
    i) The median by definition is the midpoint length.
        a. Above which 50% of the transactions are longer.
        b. And below which 50% are shorter.
    ii) The mode is the most common or most frequently occurring length.
    iii) The mean can provide a basis to calculate the total traffic volume in bytes during a given time period. We might choose to:
        i. Multiply the mean length of each type of transaction by the count of that type of transaction during the period.
        ii. Add the traffic volume for each type of transaction to get the total.

b. Do we care? Will the end conclusion of this exercise change if we use the mean length instead of the median or the mode?

c. How much is the amount of inaccuracy or imprecision introduced by the assumption that we can use one representative transaction size or length, regardless of whether it is the mean, median or mode? Let's say we developed a software simulation of the LAN traffic, where the size of each individual transaction could vary, but where either the mean, median or mode (whichever we pick as most relevant) is the same as here? The simulation is more realistic, but about how much is it likely to differ?

d. We saw earlier that the transaction volumes also vary from time period to time period? Is the amount of inaccuracy introduced by using one representative transaction volume likely to be larger or smaller than from using one representative transaction size? Why? (Hint: consider the mix of fixed-length and variable-length transactions.)

(1.4.4) Can we apply the so-called 80% - 20% rule (Pareto's principle) in this analysis, or an 85% - 15% rule?

a. Pareto's principle applies in many situations. For example, 20% of the test outsource vendors might have 80% of the market share. Here, it claims that 20% of the transactions or less account for 80% or more of the load.

b. As the following table lists 7 types of transactions, any one type accounts for

15% of the list (1 in 7).

c. Assume the accuracy required in load calculations for the initial impact assessment does not have to be better than within plus or minus 15%.

d. Can we ignore the bottom 85% (6 in 7) of the transaction types in this IIA, and still deliver a sufficiently accurate result from the calculation?

e. Better yet – do we more accurately represent the situation if we simply add another 15% to the load calculated from only the predominant transactions?

(1.1.5) Estimate the utilization of the LAN during periods of heavy load.

a. Calculate the percentage of the LAN's available capacity (10 Mbps) that will be used under heavy load.

b. Is this greater than the stated maximum threshold (35% utilization)?

(1.1.6) Are the transaction demands likely to have a material impact on the LAN, to the degree that the performance goals are unlikely to be met?

(1.4.7) If so, should the IIA recommend that the organization undertake a performance testing project? (Usually impact assessments have a broader scope and examine all areas where the impact is likely to be non-trivial, not just the LAN.)

(1.4.8) Was focusing on analyzing the LAN first in this IIA a good idea or a bad one? Why?

- Do we have an effective model of a LAN to use? (I.e., one that is realistic, reasonably accurate, and does not require onerous data gathering, assumption-making or computations.)

- Is the LAN likely to be a major constraint? Can we quickly and roughly gauge whether the main bottlenecks, if any exist for the work loads levels being used in testing, are likely to occur in other areas of the system, such as the dual T1 lines for external access?

- Are the time, cost and risk of upgrading the LAN (to 100 or 1,000 Mbps), so low that we should simply bypass this analysis, and invest what we save (by not performing the analysis) in upgrading the LAN?

(1.4.9) How much did the simplifying assumptions compromise the outcome of the IIA? In other words, what is the risk that these assumptions have led us to the wrong conclusion?

**Levels of Demand – Peak Load**

Under peak load the LAN transaction volumes are:

| **Transaction or Event Type** (Represented in the Load Mix) | **Transaction Priority** | **Throughput Load Volumes** (Transactions per Minute) | **Length – Mean** (Kilobytes) | **Length – Median** (Kilobytes) | **Length – Mode** (Kilobytes) | **Level of Confidence in the Data** |
|---|---|---|---|---|---|---|
| 1. E-mail notification of the new catalog's availability | Lo | 100 | 50 | 50 | 50 | Hi |
| 2. Request to download the table of contents page of the catalog. | Hi | 20 | 50 | 50 | 50 | Hi |
| 3. Request detail page of the catalog | Mod | 20 | 30 | 50 | 20 | Mod |
| 4. Conduct book search | Mod | 20 | 500 | 250 | 100 | Lo |
| 5. Place a new book order | Hi | 20 | 30 | 50 | 20 | Mod |
| 6. Request home page of the book club | Hi | 20 | 25 | 25 | 25 | Hi |
| 7. Other background noise (concurrent unrelated activity) | Mod | 20 | 50 | 100 | 35 | Lo |

## Assumptions

We typically need to make some simplifying assumptions in order to perform a reasonably quick assessment. In this situation, the assumptions are:

a) For now, we can ignore the question of whether the new system is able to meet its response time requirements (such as responding to internal workstation queries within 2 seconds, 90% of the time or more), and focus only on whether it can accommodate the transaction throughput.

b) All the traffic listed in the table is carried on the wired LAN, and none traverses the wireless LAN.

c) We will consider only the capacity of the wired LAN to handle the throughput, not other parts or components of the system such as the servers. We will assume that the LAN will reach its maximum capacity before a server reaches its own limits – if a bottleneck occurs before the load on the system reaches its peak, it will happen in the LAN.

d) Each interaction with the new ordering system normally includes both an input and a response. Since we expect an input to be followed fairly rapidly by a response, if the traffic volumes are low we can calculate the total traffic load on the network for each transaction by simply adding together the lengths of the input message and the response message for that transaction. (This effectively gives us one combined transaction instead of the pair.)

e) The impact of the other systems that share the same infrastructure, especially the billing and e-mail systems, is likely to be significant and thus cannot be ignored.

f) Under normal working conditions, the LAN utilization should not exceed 35%. In other words, the work load on the LAN should not exceed 35% of the theoretical maximum capacity of the LAN, which is 10 Mbps (megabits per second), when the usage is measured over any one-second interval. (When the utilization exceeds 35%, message collisions increase, the likelihood of reliable delivery falls, and transmission times become unacceptably slow.)

### INTRODUCTION TO LAN MODELS

These models are intended to provide a quick and easy way to estimate LAN utilization reasonably accurately, for capacity planning, bottleneck identification and assessing the

need for performance testing. You do not need to be a LAN administrator or network engineer to use these models. However, if terms like frame, collision and backbone are unfamiliar to you, and you would like to understand the underlying technology and the assumptions used, please read the appended section on the CD entitled: ”LAN Basics”.

**LAN Model for Low Traffic**

If the LAN bandwidth utilization is likely to be light (i.e., fall below 10%), ignore frame collisions and frame overheads and use this model. This greatly simplifies the LAN model, and the inaccuracy introduced into the calculated utilization is unlikely to exceed 5%.

However, if the calculated utilization from this model exceeds 10%, we cannot ignore collisions. We must throw away the results and re-do the calculation, using a more sophisticated model for moderate to high traffic (see later).

*Step 1:*

Determine the appropriate time period during which we should analyze the behavior of the LAN. Assume that this time period is one second. (Later, we will address how to determine if this time duration is too long, too short or about right. For now, accept that one second will do.)

*Step 2:*
a. Select the work load scenario to be analyzed and label it with an unambiguous name. Specify whether the scenario is one with typical demand or peak demand.

b. Take an inventory of the LAN transactions, and list by name the types of transactions (or frames or data packets) carried by the LAN, together with the number of times each one occurs within the time period and the size of each type of transaction. Use the attached worksheet or develop your own worksheet if you prefer.

c. Determine and state whether each type of transaction is fixed-length or variable-length. If the size varies for a particular transaction, document the mean (average), the minimum and maximum possible sizes. If known, also capture the median and mode, the standard deviation, and the nature of the distribution of transaction sizes.) Compile this information in the “List of Transaction Sizes” table in the attached worksheet.

d. If the transactions are fixed in length and the variations in volumes are known for multiple similar time periods, use the table entitled “List of Transaction Volumes” in the worksheet, not the one entitled: “List of Transaction Sizes”.

e. Often, the top 10% of the types of transactions account for 90% or more of the total traffic volume. (This is a variation of Pareto's principle.) Check if this principle applies in the situation at hand, i.e., a small number of transaction types dominate. If the principle holds, ignore the bottom 90% of the transaction types – we do not need highly precise answers.

f. Make sure your transaction sizes are consistent: do not mix bits and bytes or bytes and kilobytes (KB) without converting the sizes to a common base (1 byte = 8 bits, 1 kilobyte = 1,000 bytes). And use the same common time period when counting how many transactions occur. One second is generally an adequate time period to work with. State the dimensions and units in which your data is expressed (e.g., KB per second).

*Step 3:*

Calculate the LAN demand (and thus the expected throughput), by summing the sizes of all the transactions that the LAN needs to carry, for either a typical period of activity or for one with peak demand. Add 4% to the total size of all the transactions to allow for Ethernet overhead.

*Step 4:*

Determine the theoretical capacity of the LAN you are working with. For traditional Ethernet it is 10 Mbps or 10,000,000 bits per second, while gigabit Ethernet is 100 times faster. Convert this capacity to the same dimensions used in Step 1 above (e.g., KB per second, which is derived by dividing the bits-per-second capacity number by 8,000).

*Step 5:*

Calculate the LAN backbone utilization by dividing the demand from Step 1 by the capacity from Step 2. Multiply the answer by 100 to express it as a percentage.

*Step 6:*

Run a dimensional analysis check on the resulting LAN utilization from the calculation.

*Step 7:*

If your calculated utilization is less than 10%, the calculation is done and acceptable.

If instead your calculated utilization exceeds 10%, abandon it – it is too inaccurate. Re-work your calculations using the model for moderate to high traffic below, which accounts for collisions on the backbone.

**LAN Model for Moderate to High Traffic**

This model is more accurate and more broadly applicable than the previous one, but is more complicated are thus harder to apply correctly. We need to calculate two items: (a) the probability of a collision, and (b) the time lost in cleaning up after the collision, including re-transmission.

*Step 1:*

Assume that the appropriate time period for which we should analyze the behavior of the LAN is one second, as in Step 1 of the previous model for a LAN with low traffic.

*Step 2:*

If you have not already done so, take an inventory of the LAN transactions, as described in Step 2 of the previous model for a LAN with low traffic.

*Step 3:*

If you have not already done so, calculate the LAN demand by summing the sizes (in the same units of measure) of all the transactions that the LAN needs to carry, for either a typical time period or one with peak demand. Follow the same procedure as described previously in Step 3 for the low traffic LAN. Make sure you are working in consistent dimensions or units of measure, and state those dimensions (e.g. KB per second). Remember to add 4% to the total to allow for overhead.

*Step 4:*

If you have not already done so, determine the theoretical capacity of the LAN you are working with, as in Step 4 of the previous model.

*Step 5:*

Calculate the number of frames sent by all nodes within the same one-second time period. Assume the average frame has a 1,000-byte payload. Thus the frame count is numerically equal to the total demand from Step 3 immediately above, divided by the typical frame size (1,000 bytes).

*Step 6:*

Calculate the probability that the LAN is in a state where it is possible for a collision to occur.

a. Collisions can only occur at the beginning of each frame's transmission, and only for a brief period equal to the so-called propagation delay.
b. The delay is the elapsed time needed for any bit of information to travel from one node in the network to another. It varies and is influenced by many complex factors; for simplicity we will assume the propagation delay has a fixed duration.
c. Assume the propagation delay is equal to 100 microseconds, to simplify the arithmetic.
d. This probability of the network being in a collision-possible or collision-ready state is approximately equal to:
    i. [ The number of frames sent by all nodes during the time period ] multiplied by
    ii. [ the average propagation delay ] divided by
    iii. [ the duration of the time period ].

*Step 7:*

Calculate the likely number of actual collisions within the time period.

a. A collision will happen only if two conditions are met: the network is in a collision-possible state, a party on the network attempts to transmit a frame while the network is in that state.

b. The most likely number of collisions within a time interval is:

    Number of frames sent per node within the time period (i.e., per second)
    - x Number of connected and actively sending nodes
    - x The probability of the network being in a collision-possible state

c. This equals:

    Number of frames sent by all nodes within the time period
    - x Percentage of the time period when collisions can occur

d. Thus we do not need to know the number of connected, active nodes, including the computers, printers and other peripheral devices on the LAN.

e. In addition, the percentage of the time period when collisions can occur equals the probability of being in a collision-possible state, which we calculated in Step 6 above.

*Step 8:*

Calculate the loss per collision. To simplify the arithmetic, assume that:

a) The number of frames discarded per collision is exactly 2. (The actual number is more like 2.01, because not all collisions involve only 2 frames: about 1% of collisions are three-way.)

b) The average frame payload is 1,000 bytes.

c) There is no overhead on a frame (it actually is 38 bytes).

d) The wait time to discard corrupted frames, clear the backbone and re-transmit the discarded frames is equal to the time needed to transmit 2 frames (2,000 bytes or 16,000 bits), or 16 milliseconds at a LAN speed of 10 Mbps.

Express this collision loss not in terms of the interruption time, but in terms of the data that could have been transmitted during the interruption of service. (It is exactly equal to 2,000 bytes per collision).

Add 5% to the calculated answer, to adjust for the simplifications. The adjusted loss is 2,100 bytes per collision.

*Step 9:*

Calculate the total loss for all collisions within the one-second time period, by multiplying the likely number of collisions from Step 7 by the loss per collision from Step 8 (2,100 bytes or 16,800 bits).

*Step 10:*

Calculate the total demand on the LAN, by adding the collision losses from Step 8 to the original demand from Step 3.

*Step 11:*

Calculate the LAN backbone utilization by dividing the total demand from Step 10 by the capacity from Step 4. Multiply the answer by 100 to express it as a percentage.

*Step 12:*

Run a dimensional analysis check on the resulting LAN utilization from the calculation.

## LAN UTILIZATION WORKSHEET

Author: _____

Date: _____ Version #: _____

BACKGROUND

Name of Scenario: _____

Description of Scenario: _____

_____

Time Duration being Analyzed: _____

Units of Measure used for Volumes and Sizes: _____

Theoretical LAN Capacity: _____

Type of Load or Demand: Average ____ Peak ____

RESULTS

LAN Utilization = Total Volume / Theoretical Capacity

= _____ x _____ x 100%

= _____ %

(See the attached tables for the details used in calculating the Total Volume.)

Dimensional Analysis of the LAN Utilization Calculation

Acknowledgement that size units of measure are consistent: ____

Acknowledgement that time periods and time units of measure are consistent: ____

Acknowledgement that dimensions are consistent: ____

List of Transaction Sizes:

| Transaction Name | Variable or Fixed Length | Size Min. | Size Max. | Size Mean | Size Median | Size Mode | Units of Measure (e.g., bytes) | Volume of Trans. in Time Period | Size x Volume |
|---|---|---|---|---|---|---|---|---|---|
| 1. | | | | | | | | | |
| 2. | | | | | | | | | |
| 3. | | | | | | | | | |
| 4. | | | | | | | | | |
| 5. | | | | | | | | | |
| 6. | | | | | | | | | |
| 7. | | | | | | | | | |
| 8. | | | | | | | | | |
| 9. | | | | | | | | | |
| 10. | | | | | | | | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Total Volume | xxx | xxx | xxx | xxx | xxx | xxx | xxx | xxx | |
| Total Volume with Overhead | xxx | xxx | xxx | xxx | xxx | xxx | xxx | xxx | |

## LAN UTILIZATION WORKSHEET

List of Transaction Volumes:

| | Transaction Name | Vol. Min. | Vol. Max. | Vol. Mean | Vol. Median | Vol. Mode | Units of Measure | Size | Size x Volume |
|---|---|---|---|---|---|---|---|---|---|
| 1. | | | | | | | | | |
| 2. | | | | | | | | | |
| 3. | | | | | | | | | |
| 4. | | | | | | | | | |
| 5. | | | | | | | | | |
| 6. | | | | | | | | | |
| 7. | | | | | | | | | |
| 8. | | | | | | | | | |
| 9. | | | | | | | | | |
| 10. | | | | | | | | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Total Volume | xxx | xxx | xxx | xxx | xxx | xxx | xxx | xxx | |
| Total Volume with Overhead | xxx | xxx | xxx | xxx | xxx | xxx | xxx | xxx | |

## *Exercise 1.5: Deciding Where to Observe and What to Monitor*

### Objectives

The purpose of this exercise is to decide which behaviors of the system to observe (i.e., which performance characteristics to track during testing), where in the system (e.g., at which access points), and what data to collect.

### Instructions

(1.5.1) At what points in this architecture do you recommend that we monitor the system's performance during testing, in order to test the system's ability to meet the performance requirement described in exercise 1.3?

(1.5.2) What type of data do you recommend that we collect at each major monitoring point (e.g., end-to-end response time, throughput, bandwidth utilization, processor utilization)? For which transactions should data be collected?

(1.5.3) Is this data likely to be readily available (e.g., in server logs), or will we need to act? These actions can be invasive or not, and include loading data capture tools, inserting hardware probes (this is less common) or having people (most likely testers) observe and manually record events.

## *Exercise 1.6: Selecting the Methods of Testing*

### Objectives

The purpose of this exercise is to determine what behaviors to monitor and what characteristics to measure during testing.

### Instructions

(1.6.1) Assess the usefulness of each of the following test methods for this project, and also the relative effort needed to apply that method. In the table below, rank the usefulness and the effort on a scale of low, moderate or high. The methods are defined in the glossary on the CD.

| Test or Measurement Method | Usefulness or Value for this | Likely Effort to Apply in |
|---|---|---|

| | Project | this Situation |
|---|---|---|
| 1.0 Testing which is driven by what we want to measure. | | |
| 1.1 Response time measurement | _____ | _____ |
| 1.2 Throughput measurement | _____ | _____ |
| 1.3 Availability measurement | _____ | _____ |
| 1.4 Measurement of resource utilization | _____ | _____ |
| 1.4.1 Processor | _____ | _____ |
| 1.4.2 Main memory | _____ | _____ |
| 1.4.3 Hard drive capacity | _____ | _____ |
| 1.4.4 Network bandwidth | _____ | _____ |
| 1.4.5 Lengths of queues | _____ | _____ |
| 1.5 Error rates | | |
| 1.5.1 Inaccurate responses to application queries | _____ | _____ |
| 1.5.2 Buffer or queue overflows | | |
| 1.5.3 Page faults | _____ | _____ |
| 2.0 Testing which is based on the source or type of the load. | | |
| 2.1 Usage-based testing | _____ | _____ |
| 2.2 Standard benchmark testing | _____ | _____ |

3.0 Testing which seeks to stress the system or find its limits.

| | | |
|---|---|---|
| 3.1 Hot spot testing | _____ | _____ |
| 3.2 Spike and bounce testing | _____ | _____ |
| 3.3 Breakpoint testing | _____ | _____ |
| 3.4 Degraded mode of operation testing | _____ | _____ |

4.0 Concurrency testing

| | | |
|---|---|---|
| 4.1 Interaction / interference testing | _____ | _____ |

5.0 Risk-based testing

| | | |
|---|---|---|
| 5.1 Risk assessment | _____ | _____ |
| 5.2 Bad day testing | _____ | _____ |

## *Exercise 1.7: Determining the Test Focus and Coverage*

This exercise expands on the ideas discussed in the previous exercise, and places risk at the center of the test selection process.

**Instructions**

Perform a risk assessment in order to identify where to focus the deep, intense testing and where else to skim and test lightly, in order to conserve resources for the areas which require the deep testing. You can assess and pinpoint the risks by answering the following series of questions. You may do this exercise individually or in a team brainstorming session.

**Questions to Address – Load Demand Risks**

(1.7.1) What circumstances are likely to cause heavy demand on the system from external users (i.e., remote visitors to the web site, who are not book club employees)?

(1.7.2) Under what circumstances is heavy internal demand likely (i.e., by the book club employees)?

(1.7.3) What uses of the system are likely to consume a high level of system resources per event, regardless of how frequently the event occurs? The resource consumption should be significant for each event, not high in aggregate simply because the event happens frequently and thus the total number of events is high.

(1.7.4) What system uses are timing-critical or timing-sensitive?

(1.7.5) What uses are most popular, i.e., they frequently happen?

(1.7.6) What uses are most conspicuous, i.e., have high visibility?

(1.7.7) Based on your understanding of the system architecture and support infrastructure, where are the likely bottlenecks?

(1.7.8) What specifically is new or changed in the coming version of the system or its support infrastructure? Areas which are new or modified are more likely to have performance issues than areas which have already been running satisfactorily and have not been touched. However, if most or all of the system is new, everything is at risk and answering this question will not help.

(1.7.9) What has been your prior experience with other similar situations? Which features or systems aspects typically have encountered performance problems? If you have no experience with other similar systems, please skip this question.

(1.7.10) Are there any notably complex functions in the system, for example, in the area of exception handling?

You are now about halfway through this exercise – time for a brief break to let your brain cells cool off.

**Questions to Address – Infrastructure and Design Risks**

(1.7.11) Are there any areas in which new and immature technologies have been used, or unknown and untried methodologies?

(1.7.12) Are there any other background applications which share the same infrastructure and are expected to interfere or compete significantly for system resources (e.g., shared servers)?

(1.7.13) What is the architects' and developers' level of confidence in the system's adequacy? Do we know where in the system these people feel comfortable that performance will not be an issue, and in which areas are they nervous? I am assuming that the testers have access to the architects and designers – if not, this question and the next one may be unanswerable and thus irrelevant.

(1.7.14) What are the architects' and developers' reputations for delivering systems which fail to meet the performance goals, and their credibility in spotting potential problems? Since these people usually understand the system internals better than anyone else, their suggestions could be invaluable – but only if they know what they are talking about.

(1.7.15) What can we learn from the behavior of the existing systems that are being replaced, such as their work loads and performance characteristics? How can we apply this information in testing the new system?

(1.7.16) What areas of the system operation, if they have inadequate performance, most impact the bottom line (revenues and profits)?

(1.7.17) What combinations of the factors, which you identified by answering the previous questions, deserve a high test priority? What activities are (a) likely to happen concurrently, and (b) cause heavy load and stress on the systems?

(1.7.18) What areas of the system are low in risk and thus can be minimally tested for performance without imprudently increasing liability, in order to conserve the test resources for the areas which need heavy testing?

(1.7.19) In summary, considering the total picture, what areas should the performance test focus on? Consolidate your answers to the prior questions in this exercise to form an answer for this question, by completing a table like this:

| Area to be Tested | Likelihood or Probability of Performance Problems in this Area | Likely Cost or Consequences of the Performance Problems | Exposure (Combined Importance of Likelihood and Consequences) | Relative Ease of Testing in this Area | Test Priority for this Area |
|---|---|---|---|---|---|
| Database maintenance (updates, re-indexing, and back-ups) | High (5), because this is a highly data-dependent system with a data-centric system architecture. | High (5), because the database performance affects all parts of the system operation and is highly visible. | High (5), as this is based on the combined effect of the entries in the two columns to the left. | Moderate to high (4), as automated test cases already are available, and a tool is being acquired to run them. | High (5), as this is based on the combined effect of the entries in the two columns to the left. |
| Denial of service (DOS) attack | Low (1), because an attack is assumed to be unlikely. | High (5), because without adequate DOS controls an attack will shut the system down. | Moderate (3) | Moderate to low (2), because a large test load must be generated and delivered. | Moderate (3) |
| Incoming telephone calls to the call center, after a promotion | Moderate to high (4), but expected to decline over time as more people switch to directly ordering via the Internet. | Moderate to high (4), because people are sensitive to phone delays. Members will be irritated and business lost. However, non-telephone work is not affected. | Moderate to high (4) | Low (1), because a small army of testers are needed to manually place phone calls, or specialized, expensive call generators. | Moderate to high (4) |

## *Exercise 1.8: Calculating the Test Work Load*

### Objectives

The purpose of this exercise is to determine the mix of demands to place on the system during testing.

## Instructions

Answer these questions, based on the following set of data in Section C, Volumetric Assumptions

## Questions to Address

### A. TEST WORK LOAD VOLUMES

(1.8.1) How many web site visits or sessions are expected

a. per month?

b. in a typical hour?

(1.8.2) Can the size of the daily peak demand fluctuate from day to day?

(1.8.3) Can the location (i.e., the time of day) of the daily peak fluctuate from day to day?

(1.8.4) Should we test for the highest peak that can ever occur, or the most likely daily peak (the mode) or the average peak (the mean)?

(1.8.5) Is the most likely peak in a typical day higher than the most likely peak in a typical hour?

(1.8.6) How many web site visits or sessions are expected during the most likely peak hour of a typical day? (Hint: use your answers from the prior questions, though this carries the risk a ripple effect as errors cascade from answer to answer. Also look at the listed assumptions.)

(1.8.7) Before you try calculating your answers to the next few questions, first sketch a graph to help understand the nature of the traffic flow:

a) In this graph, set the horizontal or x axis to represent the time on the clock, with the left edge of the graph being time zero and the right edge being one hour later.

b) Set the vertical or y axis to indicate the amount of activity (e.g., the number of concurrent sessions).

c) Select any time at random within this hour of activity, as the start time for a particular session.

d) Select the length of this session at random – according to the assumptions, the average length is 15 minutes and the range is from 3 to 45 minutes.

e) Draw a horizontal bar on this graph to represent the session. The bar will stretch from the start time for the extent of the session until its end time.

f) Repeat this process until you have a dozen or so bars, and stack each new bar on top of its predecessors in the graph (with a little separation gap between each pair of bars).

g) Remember to include some bars for sessions which were already active before the hour (i.e., their start times are to the left of the zero line), and for sessions which do not end within the hour (these trail off to the right).

h) Select a random point in time and draw a vertical line at that point on the graph.

i) See how many horizontal bars intersect this line.

j) The number of concurrent sessions at this point in time is represented by the height of the pile of bars at that point, in other words, by the number of bars intersecting the vertical line.

k) Experiment with your graph. Draw another couple of vertical lines at other points, and count how many concurrent sessions are occurring at those times.

(1.8.8) What is meant by the word “concurrently”?

a. Do all on-line users have to active at any given instant, or can they be waiting for the system and vice versa?

b. What about someone who aborted but did not bother to log out?

c. Until the system times out, should we count them?

d. Do we care either way?

How about a quick break to refresh yourself before continuing! The next step will take some work.

(1.8.9) Draw a graph to represent the concurrent work activity on the web site during a typical hour. We have attached a blank starting grid after this question, on which you can draw your graph.

- Use the horizontal (x) axis to show the time, starting 15 minutes before the hour and ending 15 minutes after its end, 90 minutes in all. Use the vertical (y) axis to show the amount of activity (the number of users or sessions, both active and inactive).

- Represent each user session by a horizontal line, shown as solid for those intervals when the session is active and dotted when waiting (e.g., for think time). Each line will be a chain of interspersed solid and dotted segments.

- Stack the horizontal lines on top of each other for the same time period. For brevity, limit the horizontal lines on your graph to 10. Those will be enough to see the patterns.

- Plot the sessions (i.e., the horizontal lines) with random starting and ending points, including some before the beginning of the hour and after its end. Assume that there is on-going activity with no ramp-up or ramp-down: the web site is fully operational during the period from15 minutes before the hour begins to 15 minutes after it ends.

- See the session statistics provided below to determine the active / inactive time ratios and the session lengths.

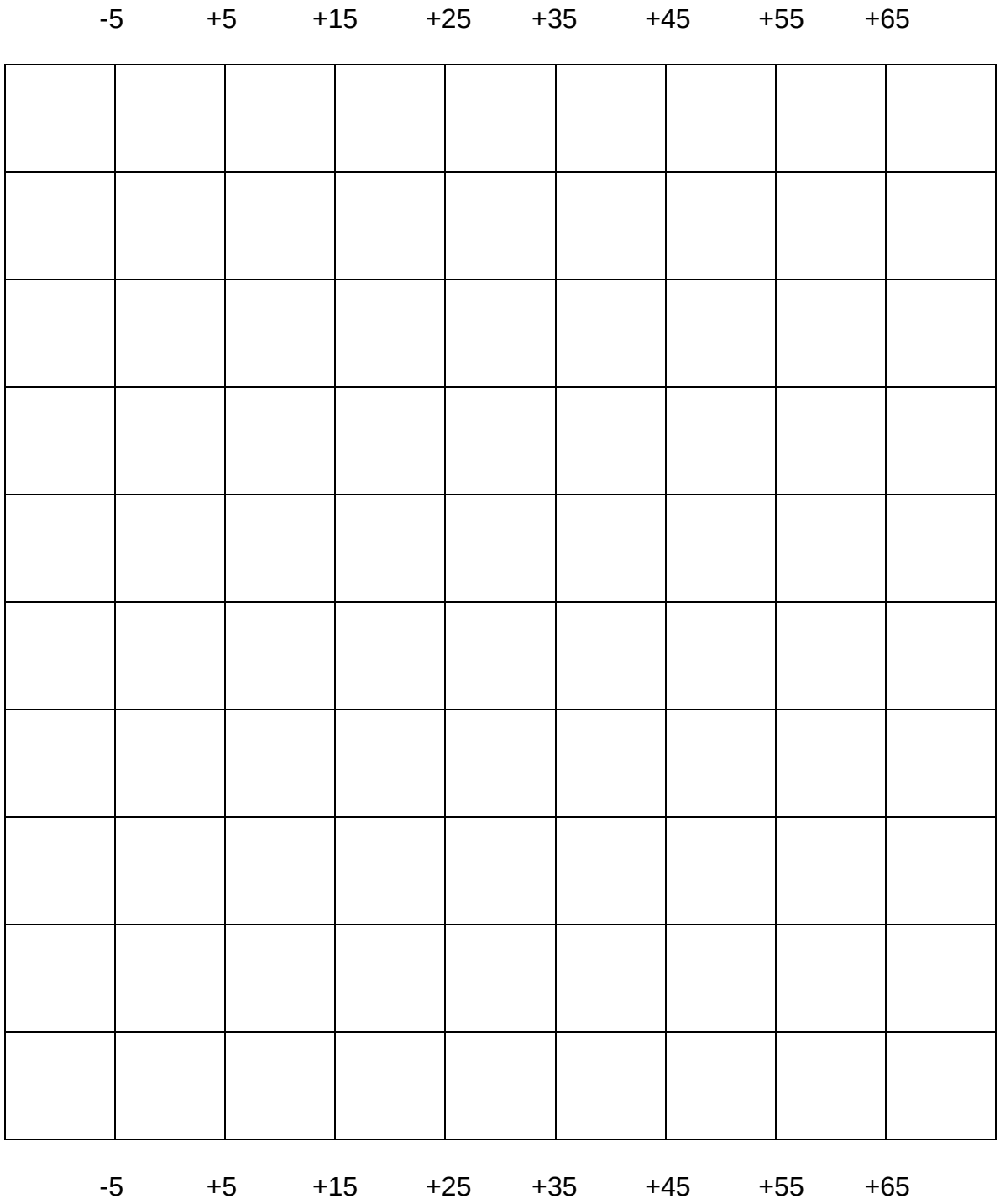
- Note that within the hour, a user may log-off and then log back on. In other words, one user could have more than one session during the hour. Another user could have no sessions (i.e., is not logged on at all). Do not include people who do not log on at all during the hour in your set of 10 users.

- At each 10-minute point on the graph, count how many concurrent user sessions there are and how many are currently active. Express these numbers as percentages of the 10 users.

- Calculate the number of active and inactive sessions at each 10-minute interval. Base these estimates on (a) the percentages you just calculated, and (b) your earlier calculation, (the answer to question 1.8.1), of many web site visits or sessions are expected in a typical hour.

- Document any significant assumptions you make that may not be obvious.

**Concurrent Work Activity**



Time in Minutes



Case Study 1.60

(4.8.10) What is the expected average number of concurrent visitors?

(4.8.11) What is the expected peak number of concurrent visitors within a typical hour?

(4.8.12) What is the expected peak number of concurrent visitors in a typical month?

(4.8.13) What is the minimum number of concurrent web site visits or sessions?

(4.8.14)How many hits and page views will happen to the home page in a typical hour?

(4.8.15) How many page views will happen to the home page in the peak hour of a typical month?

(4.8.16) Are there any inconsistencies in your calculations? Can you use dimensional analysis to find questionable answers? (This technique is explained in the glossary.)

(4.8.17) So what? How could and should we use the results of these computations in our test work load planning?

You are now about halfway through this exercise – time for a brief break to let your brain cells cool off.

## B. TEST EXECUTION LOGISTICS

(4.8.18) What should be the sample size? I.e., about how many of each significant event or transaction should we include in each test run?

(4.8.19) How should we validate and cleanse the collected data?

(4.8.20) Should we compute and use the mean, median or mode of the data observations?

(4.8.21) Approximately what should be the elapsed time for each test run? For how long do we need to observe and gather data?

(4.8.22) During this period, what can we monitor in order to ensure that the measurement process proceeds in a satisfactory manner?

(4.8.23) How do we make the decision, if necessary, on whether to continue or abort a

test run?

(4.8.24) Can we deliberately accelerate the test throughput, faster than real world rate of events occurring, in order to shorten the test duration?

a. If so, how can we adjust for this acceleration of the testing in the data analysis?

(4.8.25) Can we make the test case mix more negative (i.e., more destructive) in order to enrich the opportunities for system failure, faster than in the real world rate, in order to shorten the test duration?

b. If so, how can we adjust for this in the data analysis?

(1.8.26) What other significant assumptions besides the ones listed below, if any, have you made in this exercise?

## *Description of the Situation, Section C: Volumetric Assumptions*

Use the following data to calculate your answers to the work load questions. Please document any assumptions you make. See the glossary for terminology explanations.

**User Demand**

The 5,000 active customers will visit the web site, on average twice per month.

Another 5,000 casual visitors will access the site per month.

The volume of traffic is approximately the same from day to day.

**Session Statistics**

A site visit, or session, lasts an average of 15 minutes.

While session lengths can be shorter than a second or at the other extreme indefinitely long, 90% of sessions lengths fall in the range from 3 minutes to 45 minutes.

Each visit or session accesses and downloads the home page once on average.

For a typical user session, the ratio of active involvement with the web site (e.g., downloading a web page) to inactive (still connected, but passively reviewing or

thinking), is 10% to 90%.

**Peak Loads**

The number of sessions in the peak hour is expected to be 3 times the number in the average hour.

The peak number of concurrent users in a peak hour is expected to be 3 times the peak number in the average hour of the day.

## *Exercise 1.9: Balancing Exploratory and Structured Testing*

**Objectives**

The purpose of this exercise is to determine the right mix of exploratory and structured testing for this project. Commonly, many unknowns complicate performance testing projects and the more the unknowns the less structured and pre-planned the testing can be. Review each of the following items and assign a score for your project to that item, on a scale of 0 (the item is thoroughly understood and largely known) to 5 (the item is unknown or highly uncertain). Then sum the individual scores to obtain a total for the whole list. Typically, the more common uncertainties on performance and robustness testing projects are:

(1.9.1) How clear, complete, realistic and testable are the performance requirements? _____

(1.9.2) How well do we understand what aspects of the system's behavior are we interested in examining? In other words, what specifically do we want to measure or monitor? _____

(1.9.3) How do we analyze and derive conclusions from these measurements? _____

(1.9.4) What load(s) or mixes of demands can we place on the system while we are measuring its performance and robustness characteristics? _____

(1.9.5) How can the test equipment be set up, connected and configured? _____

(1.9.6) What and where are the vulnerabilities in the system we are testing? _____

(1.9.7) What issues, complications and hassles are we likely to encounter during this performance testing project? _____

(1.9.8) How comfortable is our team about our ability to handle these issues? _____

(1.9.9) Are the completion criteria for performance testing well defined, clear and agreed to? _____

(1.9.10) How many iterations of debugging, tuning and/or modification will the system have to go through as part of this project? _____

Total Score: _____

This number, the total score, is the approximate percentage of the testing effort on the project which should be exploratory, and the remainder of the effort will be structured. If the total score is high (above 33 out of a possible high of 50), it is important to inform the managers and clients about the inherently high risk nature of the performance testing project, and to develop contingency plans in case the project does not proceed as expected.

## *Exercise 1.10 Reviewing a Detailed Test Scenario*

### Objectives

A performance test scenario describes a particular test to run, including the mix of demands to place on the system, the test equipment and tools, what data to collect, and so on. (Some people call this a test case, test script, test condition, etc.)

The main purpose of this exercise is to determine how detailed the test planning should be for this project, and how much documentation makes sense in a performance test scenario. A secondary purpose is to learn the attributes of an effective test scenario.

### Instructions

(1.10.1) Graph the typical work flow as described in the test scenario below, on a scale with one-minute intervals across a total elapsed period of 15 minutes. In the graph, show how the load mixes change minute-by-minute and also show the flow and sequence of events.

(1.10.2) How does the situation change if we instead chose to draw the graph with one-second intervals?

(1.10.3) Are the directions in the following test scenario sufficient for a tester with average skills to perform this test successfully?

(1.10.4) Is the test scenario consistent with the performance requirement stated earlier. Should it be?

(1.10.5) List seven desirable characteristics of a performance test scenario. Consider:

- What do we want a performance test scenario to do for us?
- What do we want a performance test scenario to look like?
- What do we want a performance test scenario to contain?

## *Description of the Situation, Section D: Performance Test Scenario*

*Name of this Performance Test Scenario:*

**ANNOUNCEMENT OF A NEW "LITTLE BOOK"**

*Summary of Purpose and Intended Use:* This scenario determines whether the system can handle the demand that occurs when a new book catalog is announced and triggers a surge of new orders, or when a promotion for a particular book is distributed.

*Typical Work Flow:* A batch of 1,000 customers (20% of the 5,000 customers on the database), receive the e-mail notification of the new catalog's availability or the book promotion. These notifications are broadcast at a constant rate, and all are sent within a 10 minute period.

Of these customers, 200 (20% of the 1,000 receiving the notification) respond within an median time of 5 minutes each. (In other words, 50% respond faster than 5 minutes and 50% slower.) These people download the table of contents page of the new e-catalog that is being announced. The typical respondent then requests and reviews one more page of the catalog in the next 5 minutes. Next, 20 customers (33% of those who downloaded additional information in the form of the table of contents page, a detail page from the catalog, or book searches) each places a new book order. The median order is sent 5 minutes after. The flow of messages continues for a total elapsed time of

15 minutes after receiving the notification. Orders arriving after the first 15 minutes are ignored in this scenario, since by then the load has dropped significantly below its peak and the system behavior is no longer of interest.

Note: Depending on the organization and its established procedures, this work flow may be counted as one test case, or as a suite of 1,408 individual but related test cases (1,250 + 125 + 33 transactions, not including background noise).

**Description of the Test Scenario**

*Overall Approach:* This scenario mirrors the demands associated with the announcement of the new book and the ensuing surge of sales.

*Test Infrastructure:* The same equipment and facilities, the same support software (e.g., the OS) and database, will be employed as the ones used in the live operational environment. The scalability test approach uses infrastructure realism: the test environment will mirror the live environment, and use a copy of the full live database.

*Adjustments to Measurements*: No adjustments are needed for the differences between the test lab vs. the live environment.

*Test Equipment* (hardware, networks, databases, support software): Same as used in the live operation.

*Test Work Loads:* As described earlier, in the item entitled: Typical Work Flow

*Automated Test Scripts Used in this Scenario:*

Web site demands
- Home page downloads (mandatory)
- Book search (mandatory)
- Book query (optional for this test scenario)
- Book order (mandatory)
- Credit card authorization (mandatory)
- Query status of existing order (optional)

Internal e-mail traffic -- among all departments (mandatory)

Senior management transactions: ad-hoc query (mandatory)

## *Exercise 1.11 Designing the Test Environment*

**Objectives**

The purpose of this exercise is to analyze issues related to the test environment.

**Questions to Address**

(1.11.1) How do we generate and drive the test loads? E.g., do we perform the test manually or use automated tools?

(1.11.2) What environment (test equipment and facilities) do we need for the performance testing?

## *Exercise 1.12: Estimating the Number of Test Cycles*

Performance testing is usually highly iterative, with rapid re-tests as bottlenecks are found and resolved. This means the test facility set-up and re-run must be agile. It also means that an early if crude estimate of the number of test cycles is important – if we assume 3 cycles and the project actually requires 15, our deadline is imperiled.

(1.12.1) What are the main factors which will influence the test duration and number of cycles?

(1.12.2) Approximately how many test cycles are needed?

- To establish a working test facility, including ensuring that test equipment is installed and connected correctly, and debugging test scripts?
- To explore and build a sense of how the system works and the work load flows?
- To uncover and resolve bottlenecks?
- To build confidence that the tested system is ready to go live?

## *Exercise 1.13: Reviewing the Performance Test Plan*

(1.13.1) Audit your test plan to make sure that it will work, by addressing the questions in the checklist.

**Performance Test Plan Review Checklist**

This checklist is organized into these sections:

0. Status: Readiness of the Plan for Review
1. Experience of the Test Planners
2. Target Audience for the Test Plan
3. Commitment to the Test Project
4. Test Objectives and Scope
5. Fit of the Test Strategy to the Need
6. Test Focus and Priorities
7. Feasibility of the Test Plan
8. Validity of Assumptions
9. System Testability
10. Test Coverage
11. Test Entry Criteria
12. Test Completion Criteria
13. Test Automation Approach
14. Project Work Plan and Schedule
15. Test Plan Flexibility and Maintainability
16. Test Project Staffing
17. Roles and Responsibilities
18. Usability of the Test Plan
19. Test Environment Preparation
20. Project Tracking
21. Project Fit
22. Project Coordination
23. Test Case Design and Organization
24. Test Execution
25. Compliance with Standards
26. Testware Re-Use

A detailed list of questions follows. There is some deliberate overlap in the questions, because sometimes it is useful to ask the same question from a different perspective or in different words. Not all the questions will be relevant in any particular test plan review.

*0. Review Status*

0.1 Is the test plan reviewable? The test plan needs to be clear, readable, and understandable by the people who are reviewing it and also by the people who will be

executing the test.

0.2 Is the copy of the plan being reviewed the current and approved version rather than some earlier, obsolete one?

0.3 Who else has already reviewed this test plan? Do they whole-heartedly embrace it, or do they have reservations? Do their reservations have a foundation?

*1. Experience of the Test Planners*

1.1 Who have been the lead contributors to the test plan?

1.2 Who else have been peripherally involved in the test planning, for example, in advisory roles?

1.3 Is there a level of comfort that the people who developed the test plan have a sufficiently in-depth understanding of (a) the background subject matter, (b) the system requirements, (c) the technical environment and (d) the testing & QA techniques, in order to be able to deliver a valid plan?

*2. Target Audience*

2.1 Does the plan identify who is its intended audience – the intended readers, approvers and plan users? If it is not explicitly identified, is the audience clearly implied?

2.2 Is the audience the right one, and does the plan reflect a reasonable understanding of this audience?

*3. Commitment*

3.1 Whose commitment is needed for the test project to succeed?

3.2 Does the test project have buy-in and support it needs from these people?

3.3 If not, do the test planners have a strategy for obtaining this support?

3.4 Have the major constituencies and vested interests, who have needs and expectations from the test results, been identified in the plan? (These may or may not be the same individuals and groups whose commitment is needed for the test project to succeed.)

3.5 Have the major constituencies and vested interests been involved adequately in the test planning?

3.6 Have their needs and expectations been adequately addressed in the test plan?

3.7 Are they willing at this point to sign-off on the test plan?

*4. Test Objectives and Scope*

4.1 Is there evidence that the test planners understand the success and acceptance criteria for the overall system development or maintenance project?

4.2 Is there evidence that the test planners understand the constraints on the overall project? (These constraints include the project schedule and budget, quality practices, availability of software engineers to work with testers, etc.)

4.3 Are these success factors and constraints reflected in the test plan?

4.4 Are the test objectives clearly identified in the plan? In other words, is it clear what the test project intends to accomplish?

4.5 Are the objectives adequate, achievable and measurable?

4.6 Is the test scope clearly identified in the plan, including what will *not* be tested? Is the scope definition specific and unambiguous? Is it reasonable?

4.7 Does the scope of the planned test project encompass all the types of testing that may be needed, for example: application functionality testing, performance testing, usability testing, testing across multiple platforms, on-line help and user documentation testing, etc?

4.8 Have all the items which need to be tested (the specific features, Web pages, transactions, databases, etc.), been listed?

4.9 What allowance does the plan make for possible scope creep?

*5. Fit of the Strategy to the Need*

5.1 From the test plan, is it clear what the overall test strategy is, i.e., the approach proposed to achieve the test objectives?

5.2 Is the test strategy a coherent and practical one? Does it make sense given the circumstances?

5.3 What alternatives and trade-offs were considered for the testing?

5.4 What evidence is there that the proposed approach is the best alternative?

*6. Test Focus and Priorities*

6.1 Are the test focus and priorities clear from the plan?

6.2 Are they the right ones, given the context of this test project?

6.3 Has a risk assessment be done to identify the likely vulnerabilities of the system in live operation?

6.4 Is the risk assessment acceptable? Are there any major risks which have been overlooked or downplayed? Are there minor risks which have been overstated?

6.5 Has this risk assessment been used to focus the test effort?

6.4 Will feedback from the risk assessment be used to mitigate risk, as well as guiding the testing efforts? (These mitigation activities may be outside the scope of the testing project.)

*7. Feasibility*

7.1 Will the plan work? If the proposed approach is followed and the major steps undertaken, is there a reasonable assurance that the test objectives will be satisfied?

7.2 Is the test plan specific enough to be implemented? If we turned it over to typical testers, would they be able to proceed productively?

7.3 Is the plan overly ambitious, given the limitations of test expertise, test facilities, deadlines, and so on?

7.4 Has a risk assessment been done on the test project itself?

7.5 What are the major risks that the test project will not work as planned?

7.6 What unresolved issues are associated with the test project? What is the plan for reaching resolution?

*8. Assumptions*

8.1 Are the major assumptions underlying the plan documented, clear, reasonable and agreed to by the key parties involved?

8.2 What assumptions have been made about skills and expertise needed for testing, about resource demand, and about resource availability, as and when needed to fulfill this demand? Are these resource assumptions reasonable?

8.3 What assumptions have been made about the test facilities and tools?

8.4 What assumptions have been made about the relative cleanliness, testability and the readiness of the system to test? What assumptions have been made about the degree and quality of the testing that has already been performed in prior phases?

8.5 How can the reasonableness of the assumptions be checked?

*9. Testability*

9.1 Is the system or product specification testable, i.e., does it provide a sufficient foundation -- regardless of the form in which it may be available -- on which to determine what to test?

9.2 If not, are there other adequate sources of equivalent information about the system?

9.3 Does the test plan address how the system can be designed for testability, and the testers' role in this?

9.4 If the system design is already completed, what access points (also called hooks or probes), and what test features have been incorporated?

9.5 Does the test plan address how the testers will exploit these built-in access points and test features?

*10. Coverage*

10.1 Will test coverage be measured? If so, how?

10.2 Have coverage goals been set for the testing?

10.3 If so, does the test plan explain the rationale for the goals?

10.4 Are these coverage goals reasonable?

10.5 What is the breadth of functional coverage planned in the test, and how does this compare with the entire functional breadth of the system we are testing? If certain features are not being tested or are being lightly tested, is this justified?

10.6 If the test is for a change to an existing system, is the scope of re-testing of existing features adequate in this particular situation? Is a comprehensive regression test required after this change?

10.7 Does the depth or thoroughness of testing of each particular feature or condition (e.g., the percentage of detailed path coverage that is provided by the tests), match the risk profile of that feature? In other words, is the feature being under- or over-tested?

*11. Test Entry Criteria*

11.1 Are the criteria for the commencement of the testing clear, agreed on and reasonable? For example, are there acceptance criteria to help ensure that the system or product as delivered is actually ready to test?

*12. Test Completion Criteria*

12.1 Are the criteria for completion of the testing clear, agreed on and reasonable? When these criteria have been met, will the system really be ready to release with confidence?

12.2 What compromise pressures may be encountered during the testing, to release the product to the client(s) before it may be ready? How will these pressures be handled?

*13. Automation Approach*

13.1 Are the test tools to be used appropriate for the job – do the testers know them, and are they available to the testers and reliable?

13.2 Has adequate consideration been given to re-using existing automated test

cases?

13.3 Is the division between manual and automated testing clear from the plan, together with the rationale for choosing either manual or automated testing in the various areas of the test project?

13.4 Is the division between manual and automated testing appropriate? Would any parts of the manual testing be better if they were automated, and vice versa?

*14. Project Work Plan and Schedule*

14.1 Is there a test work plan and schedule that identifies the testing tasks, their inter-dependencies, the specific deliverables from each task, resource needs and milestone dates?

14.2 Does the test work plan unfold or flow logically?

14.3 Is the sequence in which the test activities will be done approximately the right one?

14.4 Have potential bottlenecks and constraints on the testing been identified?

14.5 How does the plan address these bottlenecks and constraints?

14.6 Are there alternate, back-up or contingency plans, if the success factors required for the proposed approach later cannot be attained?

14.7 How were the time and cost estimates for the testing calculated, and what are the estimates based on? Are they reasonable?

14.8 Is the planned amount of test activity proportionate to the amount of development and maintenance effort on the project, and to the perceived degree of risk and criticality inherent in the system?

*15. Plan Flexibility and Maintainability*

15.1 Are there contingency plans, in case events do not happen as anticipated during the testing?

15.2 If the test plan is imperfect (and they all are, to some extent), are there mechanisms to help learn from experience, revise the plan and carry on smoothly with

the revised version during the test execution process?

15.3 Does the plan identify the individual(s) who are authorized to approve changes to the system functionality and scope?

*16. Project Staffing*

16.1 Does the plan clearly state what skills and expertise are needed?

16.2 Are people with sufficient skills available when the plan says they will be needed?

16.3 Has adequate consideration been given to hiring consultants, or out-sourcing the portions of the testing where internal resources are lacking?

16.4 Does the plan identify the possible points of contention with other projects which will be competing for the same scarce people?

16.5 How does the test plan propose these resource conflicts be resolved?

*17. Roles and Responsibilities*

17.1 For the groups and individuals who will be involved in the testing, have their respective roles and responsibilities been defined or at least outlined?

17.2 Have the people who will follow and execute the test, actually reviewed what is going to be expected of them? Do they understand the specific tasks, work load, working conditions, time demands, unusual demands such as weekend overtime (if any), deadlines to be met, reporting relationships during the test project, and who buys the doughnuts? (It's essential to feed the testers.)

*18. Usability of the Test Plan*

18.1 Who are the people who will have to follow this plan and execute the test?

18.2 What are their characteristics, availability and skills?

18.3 What do they need to know in order to test successfully, and is this provided by the test plan?

18.4 Is it clear how the people will actually use the test plan? Why, when and how in their work?

18.5 Have these people reviewed the test plan?

18.6 If so, what is their opinion? Does it fill their needs and is it usable from their perspective?

18.7 Is the plan specific enough to be implemented. If we turn it over to the people who will do the work, can they proceed effectively?

*19. Test Environment Preparation*

19.1 What facilities are needed for the test?

19.2 How well do they represent reality, i.e., mimic the live operation?

19.3 Are the requested test facilities already available?

19.4 If not, does the test plan describe how they will be acquired, built or borrowed?

19.5 Is the acquisition of the facilities realistic?

19.6 Is the plan for setting up, checking out and maintaining the test environment sufficient and practical?

19.7 Are the any known limitations of the test environment? If so, how does the test plan adjust for them?

*20. Project Tracking*

20.1 During test execution, how will the test project status be monitored?

20.2 How will the testers assess whether they are on track, or make mid-course corrections if needed?

20.3 How will the overall test effort be managed and progress reported?

20.4 Are these questions addressed in the test plan?

*21. Project Fit*

21.1 Does the test project fit suitably within the context of the overall system

development or maintenance project?

21.2 Does the testing fit suitably within the overall corporate culture, “the way things are done around here”?

*22. Project Coordination*

22.1 Does the test plan address how the test execution effort will be coordinated with other related testing activities, such as problem reporting, follow-up and re-testing?

22.2 How will the testing activities be coordinated with the version control, change and release management processes?

*23. Test Case Design and Organization*

23.1 Does the test plan identify the test case design techniques which will be used in this project?

23.2 If so, are the selected techniques appropriate in this situation?

If the test plan contains test cases:

23.3 Are the test cases traceable back to the features or specification(s) – i.e., is there an easy-to-use cross-reference among the test cases and the features we are testing?

23.4 Are the individual test cases feasible and practical? At the least, a representative handful of the test cases in the test plan should be selected and reviewed for practicality.

23.5 Do the test cases comply with the expected standard format for test cases?

23.6 Are the test cases cross-referenced back to the system requirements, features and system versions?

See also the test case design checklist in my book entitled: “Developing Effective Software Test Cases”.

*24. Test Execution*

24.1 Does the planned sequence or flow of the test case execution proceed from the most critical to the least critical?

24.2 Does the test plan describe what test support processes will be used, e.g, for problem reporting and resolution? Are these processes appropriate for the project?

*25. Compliance with Standards*

25.1 Does the test plan comply with the organization's expected standard format for test plans?

25.2 Does the plan comply with the organization's guidelines for the use of test facilities, tools and test procedures?

25.2 Does the test plan use the standard terminology which is used in the organization? Does it contain or refer to a glossary of terms?

25.3 Has the test plan been placed under version control?

25.4 Have copies of the test plan been filed with the appropriate groups (e.g., the test librarian)?

*26. Testware Re-Use*

26.1 Is the test plan, or at least its relevant portions, designed to be re-usable in future testing projects?

Is this review checklist longer than your test plan itself? (While this question is meant as a joke, at least one organization has stated that its test plans cannot exceed two pages.)

# II. The Full Case Study: Understanding the Situation

The exercise below is the first in a series where you, working individually or in a small team, will develop a performance testing strategy. If you have not already done so, read the earlier “Introduction to the Case Study” before continuing.

## *Exercise 2.1: Reviewing the Proposed Testing Objectives*

### Introduction

Your purpose in this first exercise is to understand a typical business situation, analyze the pertinent issues and consider what the performance testing objectives should be.

### Instructions

First, read the background to the case study in the attached Description of the Situation, Section 2.A only. (You do not need to read Sections 2.B and later for this exercise.) In class, we won’t take the time to first fully absorb and intensely critique the description, as we want to start the group discussion of the issues as soon as we have a sense of the situation.

You may say: “Why should I have to read this? The only thing I ever read is TV Guide.” Actually, this reading is important. Testing is context-specific, and we can talk endlessly about test strategy, but there is no substitute for actually getting in there and doing it ourselves. This background reading provides the context – it describes a typical challenging situation you are likely to encounter in your job.

Second, based on the background (i.e., the description of the situation), answer the questions listed below. Each answer need be no more than a few lines long. In the time available for this exercise you may not be able to complete all the questions.

## *II. The Full Case Study: Understanding the Situation*

The intention here is not to rush so that all questions are covered, but to think about the issues in developing a performance test strategy. These questions are tough. We do not expect perfect answers, but we would like your best thinking. If you become bogged down on a question, however, it is not worth agonizing over, so move on after a few minutes to the next question. Later, Part 2 of the case study provides suggested answers to these questions.

**Questions to Address**

Note that we are not looking for polished and detailed answers at this time, just an initial sketch of your thoughts, ready for discussion with the others in the class.

(2.1.1) What do we want to accomplish with this performance testing project? This question is typical of what you will face on the job, but it may appear daunting at first sight. Hint: one way of tackling this question (though not the only way), is to break it down into a series of a few other questions, such as:

- Why are we doing this test?
- Who do we need to satisfy?
    - Who are the vested interests: who have needs and expectations of the system performance and of the test results? These individuals or groups sometimes are called stakeholders or constituencies.
    - Note that there are usually significant constituencies beyond the immediate user community, including some who use the system little if at all.
- What is the relative importance and influence of each of these constituencies? Presumably, this ranking sets the priorities for the amount of attention and service each receives.
- What do they want to know, in terms of the uncertainties to be resolved and the issues clarified by the performance test?
- What SHOULD they want to know?
- How will they use the results of the performance test?

(2.1.2) Can the testing be avoided? Some of the senior managers are skeptical about

the performance testing project, and believe in the "fat server" approach instead. How would you respond to the fat-server suggestion that we skip the performance testing, save the time and money needed for this effort, and simply "beef up" the equipment and communications bandwidth as and when necessary in live operation?

(2.1.3) Can we define system performance goals independently of the technical environment, specifically how the system is implemented and what infrastructure it uses?

(2.1.4) Do we need an initial impact assessment (IIA) for this project? An IIA assesses the need for a performance test quickly and early, and gives a first impression of its likely focus, scope and size, based on the limited early information available about the situation. Its purpose is to identify projects where a performance test is needed and the test time and cost is justified, versus ones where the performance issues do not justify testing. (Though Appendix B describes the initial impact assessment in some detail, you will not need to review this appendix yet.)

(2.1.5) Do we need to take a baseline in order to assess the testing objectives? The term "baseline" is defined in Appendix A.

(2.1.6) A set of business objectives are included in the description of the situation. Which of these business objectives can reasonably be addressed in a performance testing project? Which cannot?

(2.1.7) Performance goals are stated in the description of the situation (Section 2.A). Overall, are these performance goals for the system (i) relevant and significant, and (ii) realistic (i.e., probably feasible to attain)? Has any major performance-related goal been omitted?

(2.1.8) Performance testing objectives also are stated in the description of the situation. Which of these testing objectives can be directly linked back to one or more specific *business* objectives? Which cannot?

(2.1.9) Skip this question if you wish – it is included to reinforce the lessons learned from answering the last two questions. Which of these testing objectives can be directly linked back to one or more specific performance goals? Which cannot? (A performance goal differs from a testing objective. An example of a testing objective is to evaluate whether response time is adequate, by comparing the

goal (a response time target) with the measured response time. The goal is the response time that the system is expected to meet under certain circumstances. It is difficult to evaluate if performance is satisfactory without having reasonably specific goals.)

(2.1.10) Are these testing objectives measurable, specific and objective enough so that we can evaluate if they have been accomplished? (The specificity of the testing objectives depends on the specificity of the performance goals. If you felt that the stated performance goals are not specific enough, assume for the moment that these goals have been revised and now are realistic and measurable. In other words, answer this question as if the revised goals can be used to evaluate the system's performance.) Realistically, how specific can the testing objectives be at this time, given the uncertainties and limits on our knowledge?

(2.1.11) In sum, are the stated testing objectives and performance goals appropriate? If not, what should they be?

(2.1.12) How will the constraints (see the section entitled: "Testing Constraints") affect the performance testing project?

## *Follow-Up: Team Discussion of the Testing Objectives*

(Allow 60 to 90 minutes for this exercise when you are working in a team as part of a class, or outside class if you have a group of peers who have worked through exercise 2.1 and are ready to discuss it.)

The purpose of this exercise is to compare your answers to Exercise 2.1 with others', and especially to be exposed to the thinking of people from different backgrounds and with different perspectives.

**Instructions**

If you have not previously organized teams, form a team with two to three other classmates for this exercise. Find a comfortable place to gather around and work together. Together with your teammates, compare your answers to the previous questions in Exercise 2.1. The intention in comparing answers is not necessarily to reach consensus, though that's fine, but to obtain a deeper appreciation of the issues by seeing others' perspectives.

Allow only about 10 minutes to discuss each question, though your team can choose to take longer if the extended discussion is useful. (You will get through fewer questions in

the time allotted if you take longer per question. Some of these questions could take 3 weeks each to discuss, but we only have limited time in the classroom.) It is OK if you do not get through all the questions in the time allotted, but move on to the next question if you feel that you are becoming bogged down on any one question. At the end of the exercise, be prepared to discuss and justify your team's answers with the class.

## *Description of the Situation, Section 2.A: The Business Context*

### A.1 OVERVIEW

The following sections provide the background information needed for the exercises. These sections are labeled:

**A.1.1 Context**

- A.2 The Business Background
- A.3 Your Responsibilities
- A.4 Basic Functions of the System
- A.5 Interfacing Systems
- A.6 Business Operations and Processes
- A.7 System Work Flows

**A.1.2 Objectives**

- A.8 Business Objectives for the System
- A.9 Performance Goals
- A.10 System Requirements which Influence Performance
- A.11 Assessment of the Current Performance and Robustness
- A.12 Performance Testing Objectives
- A.13 Critical Success Factors

Later, more information about the situation is provided in other follow-on sections (the Description of the Situation, Sections 2.B, 2.C, 2.D and 2.E), but this information is not needed for the first few exercises.

### A.2 THE BUSINESS BACKGROUND

Testing Books (TB) is a book club which specializes in selling testing and quality assurance books. The official company slogan is: "Test Geeks 'R Us" and the web site is testingbooks.com. The senior managers of the book club describe the business as thriving, growing and profitable. They believe the book club has a core of loyal fans that

prefer its services to generalists like Barnes & Noble and Amazon, and they want to build on this success by providing even better service and more competitive prices.

To support this business goal, TB is in the process of building a comprehensive new information system to support its core business operation, which is the ordering and shipping of books to its members. This new system will essentially replace the existing automated and manual systems which the book club uses for ordering and shipping books, and also replace an existing web site which is not considered to be very effective.

### A.3 YOUR RESPONSIBILITIES

Your job will be to test the performance of this new system, because its performance is critical to the success of the system and thus to the health of the book club. The scope of this testing project includes measuring and evaluating the system response time, throughput, error rates, resource utilization, scalability and ability to handle peak loads. Congratulations on your new assignment (or condolences). You will report directly to the vice president of information systems for the purposes of this project.

Your immediate assignment is to draft a high-level strategy which describes how you will proceed and the approach you recommend for this testing project. You will present and discuss this performance test strategy with the senior managers of Testing Books next week. In this presentation, they will want to know how you will test the performance of the system -- not the details as yet, but your overall approach. The managers expect an insightful, cogent analysis, and they are confident that what you say next week will be well thought through, organized and pertinent. (No pressure here at all!)

### A.4 THE BASIC FUNCTIONS OF THE SYSTEM

The new system processes book orders: it provides order entry and order fulfillment capabilities. With this system, customers can order books directly from the web site and book club employees can enter orders through an Intranet (an internal client/server network which uses virtually the same interface as the external customers see). The system also manages the fulfillment of these orders. In all, it will support the following business activities:

- Ordering of books from the book club.
- Picking, packing and shipping of books to members in fulfillment of orders.

- Answering queries on the status of memberships, orders and shipments.
- Publication of electronic and printed catalogs which show what books are available to order.
- Broadcasting of special offers and promotions.
- Reporting of the management information needed by the book club executives to run the business.

This management information, which is automatically generated by the system, includes cash flow projections, order volumes, trends (which books are selling swiftly or slowly), order backlogs, the turnaround times to fulfill orders, book inventory levels, etc. It is used by the senior managers who are running the business. While the volume of transactions in this category is likely to be relatively low, it is important to be able to provide timely answers – executives do not like to wait. Much of the management information, though not all, does not have to be more up-to-date than 12 hours ago.

This section does not list all the business activities and the system functions that support them, just the major ones. The complete list of functions that the system is expected to provide is listed in the system requirements document, and a related set of use cases describe the services provided and how the functions should work. The system requirements and the use cases are not attached to this exercise, but this overview will provide enough information for you to do the exercises.

To realistically measure performance, testers need to know how the system works. At this stage, you may have questions about the functionality and work flow, such as: “How does one become a book club member?”, “Can non-members access all parts of the web site?”, “Is this particular function even on the web?”, and “How does this system tie into other systems?” A later section, A.12 System Work Flows, explains the main system-user interactions.

### A.5 THE INTERFACING SYSTEMS

In addition to the book ordering system, the plan is to migrate other existing application systems to the same technical environment (the servers, databases and networks), and run them in this environment also. The main systems which share the same resources are:

- Member Billing: generates invoices for books ordered from the book club by

members, and tracks shipping and payment status of these orders.

- Publisher Ordering: generates orders for books ordered by the book club from publishers, and tracks their delivery status.

- Communications: handles internal and external e-mail messages.

### A.6 THE BUSINESS OPERATIONS AND PROCESSES

The book club business is organized into five main groups: (a) senior management, (b) the customer service group, (c) the catalog publishing group, (d) the warehouse distribution group, and (e) the information systems group. Each group has an assigned set of responsibilities, and there is little or no cross-over of tasks among the five groups.

*The customer service group works directly with the book club members*. Members can access the club's web site or telephone (speaking to a book club employee), in order to place orders, query order status, make complaints, and change information about their memberships. The web and phone orders and member data changes are processed while the member is on-line or is on the phone. The customer service group is intended to have approximately 100 personal computers (PCs) for its 100 employees. (The entire business has approximately 300 employees.)

*The catalog publishing group chooses the books for the club to offer*, collect\s book reviews and prepares a monthly catalog of available books, which is either printed and mailed or is distributed via the Internet to the members. (These monthly catalogs do not contain the full list of books available for sale, just the most topical ones.) The catalog publishing group is intended to have 25 PCs to support their work. The entire staff of the catalog publishing group is located at the remote satellite office. (All the other business groups are located in the headquarters building.)

*The warehouse group distributes books to the club members*, either from a member order or as the default monthly selection. A copy of the current monthly book selection is automatically sent to each active member, unless he or she explicitly informs the club that he does not want this book. Stocks of high-demand titles are maintained at the warehouse for filling customer orders. When the supply of any title becomes depleted the warehouse issues a request to replenish the stock to the publisher. A book shipment to a member can originate only from the warehouse -- other system users cannot authorize shipments. Personnel at the warehouse pack and ship approximately 2,000 packages containing about 2,250 books to members in a typical day, or about 50,000 books a month (working 22.5 days per month on average). The warehouse will have 50

PCs, and 25 handheld wireless devices (also called PDAs or personal digital assistants) to guide workers who pick books off shelves and pack them.

*The senior managers make ad-hoc queries and receive on-line status reports and graphs* which help them to manage the business. They also expect to receive regularly printed-out monitoring reports from the new system. There are 25 PCs planned for this group's use, including support staff such as the administrative assistants to the managers.

*The information systems group supports and maintains the computer systems* and will be expected to support the new system. This group will have 25 PCs including a test lab.

### A.7 SYSTEM WORK FLOWS

The main interactions among the system and its users are as follows.

People can order books either through the web site, or by phone or mail request. Using the web site, interested parties can search for books by topic, author, etc., query the availability and price of a book, and query the status of an order in progress. If they choose, people can enroll as book club members via the web site, phone or mail. A person does not have to become a member to order a book, but members get special privileges such as early notification of sales and occasional discounts. There are no membership fees. Book club members receive monthly catalogs and promotions, either electronically or in printed form.

An order can contain books from various publishers, with different quantities of each book ordered. Orders can be modified or cancelled at any time up until the day of shipment. If a book is not in stock, the system informs the person (or the internal book club employee) and asks if he or she wants to place a backorder.

A database record is created for every new order, and this record is updated to track the progression of the order through to fulfillment. The ordering system maintains records on the book inventory as well as on orders. When an order is entered, the system generates directives to the warehouse staff to pick, pack and ship the order, as well as printing the paperwork needed to ship the order. The system provides the capability to query and update the inventory of books on hand, and automatically decrements the inventory as books are shipped. Ordering books from publishers and book distributors is not part of this system.

The ordering system itself does not generate bills or process payments, but triggers these actions by the separate billing system. People usually pay by credit card, and members with acceptable credit history can be billed for payment within 30 days. To collect payment, the ordering system sends a transaction to the separate billing system. In the event of returned books, the billing system issues refunds but these do not flow through the ordering system. The system also provides the capability for the catalog publishing group to update book information such as descriptions and reviews, and to compose and publish the catalogs.

### A.8 THE BUSINESS OBJECTIVES FOR THE SYSTEM

The overall business goals of the senior managers are to (a) grow revenues, (b) improve profitability, and (c) increase member satisfaction. They have agreed to fund the new system, on the understanding that it will facilitate meeting these goals.

The specific business objectives for the new system are as follows:

a) Support the operations of the book club, by providing the order entry, fulfillment and related features. (These features are described in more detail in the system requirements documents.)

b) Double the volumes of orders entered directly by customers via the web within a year, and increase their percentages to 80% or more of all orders within three years (i.e., reduce the number of mail and telephone orders processed by employees to 20% or less of the total).

c) Improve productivity in the customer service and warehouse distribution groups by 25%. Productivity in these areas is measured by the number of orders processed or packages shipped per employee per hour. These measures of productivity are not affected by the shifting ratios of orders entered by customers via the web versus by employees.

d) Distribute 95% of all books within 15 working days from the date of an order and at the least available shipping cost. Here the word “distribute” means that the order is received by the customer.

e) Distribute books and packages with 98% accuracy, i.e., only 2% of the books or less should be returned because the wrong book was shipped to a member or because the member’s address was incorrect.

f) Answer 95% of customer telephone calls (e.g., queries about order status), within 3 minutes.

g) Support the projected growth in membership for the next 5 years. Today's system configuration does not have to support the load predicted for 5 years in the future, but the system must be upgradeable to meet the projected demand.

   - All systems are upgradeable -- if there are no time and cost limits. The requirements here are that upgrades:
     - Decrease or at least do not increase the per-item processing costs (e.g., per order).
     - Can be implemented well before the current capacity is consumed, i.e., the elapsed time from when a need to upgrade is identified to its implementation is short enough to avoid capacity overload.
   - The projected growth is described later in this document (in the Description of the Situation, Section 2.D).

### A.9 THE PERFORMANCE GOALS

In order to fulfill the business objectives, the managers have stated that the system must meet these performance goals:

a) Response times must be satisfactory when the system is operating under both average and peak loads.

b) The system must operate correctly when accessed simultaneously by multiple users.

c) The system must be able to handle heavy loads.

d) Satisfactory performance and reliability levels must be maintained over an extended period of use (24x365 operations).

e) The system availability (uptime) must be adequate in live operation.

f) The entire system must be tuned optimally in order to efficiently utilize the computing resources.

g) The system must degrade gracefully, not fail catastrophically, when it is pushed

up to, and beyond, it's planned maximum capacity.

h) The system must be scalable, so it can be upgraded in the future to accommodate the projected growth, without creating per-item processing costs.

### A.10 SYSTEM CONSTRAINTS

System requirements in areas like usability, security and maintainability usually are not performance requirements per se but they often are performance-related. Testing that these other requirements have been met is outside the scope of the performance testing project. Nevertheless, these system requirements may significantly influence performance. If the performance goals are met during testing, but without these requirements being satisfied, then the executives will consider the performance test results invalid. The performance-related system requirements are:

#### A.10.1 Usability

a) The system must be user-friendly, so that visitors to the web site will be encouraged to browse and order books.

b) The system features must have a consistent format and methods of navigation.

c) It should use high quality, high definition graphs and video and audio clips.

d) The web site must be viewable on all popular browsers and platforms.

e) Users should be able to set their own preferences, e.g., identify their favorite topics and ask to be notified about future new books on those topics.

#### A.10.2 Data Availability and Integrity

a) Any member must be able to retrieve his or her full history, if desired, via the web site. An analysis has found that 2% of members have each placed over 100 orders in the last 2 years – and could request their full histories. The oldest history still kept on file dates from the founding of the book club 20 years ago.

b) The system must be able to provide ad hoc management reports on demand. An example of a recent ad hoc report: a manager requested a correlation of the increases in books sold after price reductions designed to move obsolete inventory, with the amounts of the reductions.

c) The database must be continually updated in order to provide the latest information. Is the information on the mirror servers being updated often enough, as per the business requirements? Most data does not have to be more current than within the last 12 hours, but selected data should be up-to-date within seconds, such as changes to orders which currently are being picked and shipped.

d) Referential integrity (RI) must be enabled for all data which is deemed critical. RI is defined in Appendix A. It will be enabled on all currently active orders – those being picked and shipped within the next hour – and the book inventory needed for those orders. (OK, so the DBA is a little unconventional.)

e) The call center must be able to replay any recorded voice message on demand, for up to three days.

### A.10.3 Security

a) The system must allow new members to establish secure accounts' from which to order books and authorize payments, etc.

b) Members' financial information and payment transactions must be encrypted using an approved standard encryption algorithm.

c) All of a member's data must be password protected, and no other member's data can be accessed with the first member's ID and password.

d) The system must be secure from hackers.

### A.10.4 Maintainability

a) The system must be easy to maintain.

b) The source code must be under version control.

c) The source code must comply with programming standards (which may have performance implications), be modular, decoupled with controlled interfaces among software components, and be documented, visible in its actions and traceable.

d) Software components must be designed for re-use.

e) The design should facilitate quick, inexpensive modifications with minimal likelihood of introducing unintended side effects.

### A.11 Assessment of the Current System Performance and Robustness

The current systems are being replaced because their functionality, usability, security, data integrity, performance, robustness and scalability all can be improved, and for costs less than the likely benefits.

In general the current system response times and availability are adequate under the current load levels. But limited load tests using the existing systems have shown that these response times and availability cannot be sustained as the work volumes grow to the levels that the managers want to reach.

There is one main exception to the general statement that current performance is OK under normal loads. That is the search function, which receives frequent complaints about its speed and ease of use. The database administrator (DBA) maintaining the current databases believes these complaints will not be erased without a major database re-design. Another feature of the current systems, which is performed infrequently but is notorious for its lethargy, is the sales tax audit reporting.

In addition, the planned broadening of the system features and the upgrades needed to improve the system usability, security, data integrity and scalability (if the current system was upgraded rather than replaced), are expected to degrade performance.

### A.12 The Performance Testing Objectives

The purpose of this performance testing project, as outlined by the senior managers, is to assess the adequacy of the system's performance in live operation. The questions they want answered by the performance testing project are:

a) Are response times satisfactory when the system is operating under realistic loads?

b) Can the system accommodate the anticipated demand, i.e., the traffic volume?

c) Does the system operate correctly when accessed simultaneously by large numbers of users? (Problems such as features interfering with each other,

database locking, resource contention and transaction priorities may need to be considered here.)

d) How well does the system handle heavy and peak loads? (We cannot assume that the performance degradation from adding additional users or performing extra work is non-existent, or gradual or linear. A significant increase in response time may occur when only a few more users are added or the work load increases by a small increment.)

e) Does the system maintain performance and reliability levels over extended periods of use, for example, after weeks of running 24x7? (Insidious problems such as memory leaks will not reveal themselves in short run tests or by testing with a small number of users. Such problems usually lead to performance degradation and eventually to system failures.)

f) Will the system availability (uptime) be adequate in live operation? Apart from the planned downtime for maintenance, is there reasonable confidence that the unplanned downtime will be held to acceptable levels?

g) Is the system right-sized and tuned optimally?

h) Does the system degrade gracefully, or fail catastrophically, when it is pushed up to, and beyond, it's planned maximum capacity?

i) Is the system scalable: can it be upgraded in the future to accommodate the projected growth over the next 3 to 5 years, without major software re-writes or a major re-structuring or conversion of the database?

### A.13 TRADE-OFFS

There are trade-offs among the performance characteristics of any system, and usually complex interrelationships. Response time tends to degrade as demand volumes increase. Availability tends to fall as systems are right-sized, as there is less spare capacity in reserve for surges or spikes in demand.
To guide decision-making about trade-offs, the senior mangers have declared these priorities:

- Throughput takes precedence over responsiveness. While both are important, if forced to choose between throughput and response time the managers would rather slow the performance for many people than turn some away.

- The managers feel that when the system is down they effectively are out of business. Availability takes precedence over efficiency. While they do not wish the system to be grossly over-engineered and thus inefficient and unduly expensive, they fear this outcome less than being down.

## *Exercise 2.2: Modeling the Architecture*

### Introduction

To test a system adequately requires a basic sense of what it does and how it works. An appropriate model helps us to understand the system and its environment, and gaining that understanding is the purpose of this exercise.

Developing a workable, realistic model of the live environment is a key to effective performance testing and related activities like capacity planning, diagnostics and tuning, and monitoring service levels. A model can help us analyze the system's behavior, design the system for testability, speculate intelligently about resource utilization and bottlenecks, decide what demands to place on the system during testing, and decide where to monitor performance and what data to collect.

Our feel for the environment also will be important in designing the performance test facilities and developing a test automation strategy, but these issues are addressed later and not in this exercise.

You may ask why you have to develop the system architecture diagrams. These are often provided by the architects, and if they are unavailable we could simply request that the architects develop them. However, there is an *important benefit* to testers sketching the diagrams: this act brings us close to the design. We are encouraged to think through how the design works or is intended to work, and we are less likely to uncritically accept assumptions, supposed facts and proposed solutions.

### Instructions

(2.2.1) Review the system architecture (the blueprint for the infrastructure), which is described below in the Description of the Situation, Section 2.B. Because this section has a great deal of information to digest in one reading, it is better to begin outlining your answers to the questions as you go and before you have the full picture. I suggest you read only sections B.1 through B.5 for exercises 2.2.2.1 through 2.2.3.4. Then read the remainder of the sections (B.6 through B.10) before revising and completing your

answers to these questions.

(2.2.2) Please answer these questions, based on your reading of B.1 through B.5:

(2.2.2.1) Which parts of the information presented in Section B (more precisely, in B.1 through B.5), are irrelevant to gaining an initial, broad-but-not-deep sense of the situation for testing purposes, and thus their possible inaccuracies can be ignored during this exercise?

(2.2.2.2) What unclear statements, omissions, factual errors, unstated or dubious assumptions, ambiguities or inconsistencies did you find? Please list them.

(2.2.2.3) Which of these must be resolved before you map the system architecture and finalize your logical architecture diagrams (as requested below in 2.2.3)?

(2.2.2.4) What if these points cannot be resolved anytime soon? Can you proceed with developing your test strategy without resolving them? If so, how?

(2.2.2.5) What missing information would you like to know about the environment and infrastructure but is not available?

(2.2.2.6) What assumptions have you made about the missing information? Please list them.

(2.2.2.7) How sensitive is your answer to these assumptions? (If your assumptions varied, for example, how would that affect your diagrams below? I have included this question here because it has a natural link to the prior question. But this question is hard to answer before you try to draw your diagrams and see where the obstacles are. I suggest you make a note to return to this question later.)

(2.2.3) To familiarize yourself with the environment, sketch the *logical* system architecture based on the following description (initially using sections B.1 through B.5 only from that description).

There are four main types of logical architecture diagrams: user-oriented, function-oriented, geographically-oriented and device-oriented. Each provides a different perspective of the same situation, and all can represent and map into the same physical implementation. See Appendix J for examples of typical architecture diagrams.

## *II. The Full Case Study: Understanding the Situation*

Learning and mapping the architecture are best done iteratively – rather than trying to understand everything up front and then drawing a full, unblemished diagram the first time, most of us work better if we sketch as we go, erasing and erasing the diagram as we learn more. I deliberately have limited this exercise to using the data in B.1 through B.6. Later, we will return and revise the diagrams to incorporate more data, as presented in sections B.6 through B.10)

(2.2.3.1) Sketch a one-page user-oriented architecture.

- In this view, the intent is to show the facilities by their deployment to user groups: which user departments utilize which capabilities and application functionality.

- In your sketch, represent each major user group by a dedicated logical server. A simple square rectangle, labeled with the department's or user group's name, will suffice to represent this.

- Link the servers together to reflect how they are likely to be connected in operation. Showing that devices can communicate is important, but the routing can be inexact at this stage – for example, a star network configuration in this logical view could actually be implemented as a ring. Also show the major links to the external world.

- Show a summary of the peripheral devices connected to each server, such as the hand-held wireless devices. A summary is sufficient – there is no need to draw dozens or hundreds of icons, one for each individual peripheral. Include only the peripherals that are user-visible and user-significant: if most users are unaware that a certain peripheral exists, do not place it in the diagram.

(2.2.3.2) Sketch a function-oriented architecture.

- Sketch a one-page view of the front-end functions, as described later.

- Sketch a separate one-page view of the back-end functions, as described later.

- In this view, ignore for the moment the clustering, hot back-up, data mirroring and load balancing. Showing the duplicate load-sharing devices

tends to clutter the picture. These can be taken care of by adding a brief footnote to the diagram, stating that fail-over features and redundant devices will be included.

- Ignore questions of ownership or geography, just lay out the major functions provided (web services, database access, printing, etc.)

(2.2.3.3) Sketch a one-page geographically-oriented architecture. Superimpose your view on a geographical map, in this case a map of the U.S.

(2.2.3.4) Sketch a device-oriented architecture.

- Depending on how much detail you choose to include, this may spread over a few pages. The pages should fit together to provide a coherent overall view.

- Since some physical design decisions are not yet final, you may need to either ignore details or find a way to show the uncertainties in your diagram.

- Because the detailed data is usually voluminous and can lead to a cluttered, confusing diagram, do not attempt to capture all the physical device data in one page. This information includes each device's brand and model number, processor speed, amount of semiconductor memory, etc. Instead, find a reader-friendly way to show the most important information directly in the diagram, and provide references to the remainder off-diagram.

While these diagrams are intended to present the logical architecture rather than the physical, your diagrams should include what we know about the likely physical implementation at this stage, providing the information is pertinent to performance testing. The higher the level of abstraction, the more caution is needed to avoid performance surprises. This means your diagrams should include the main hardware platforms and devices (servers, routers, switches and controllers), databases and networks, their main interconnections and the connections to the outside world.

The thought content of the diagrams is more important than the symbols and conventions you use, and any reasonably clear set of graphics convention is acceptable. You do not have to make your diagram comply with standards like UML.

## *II. The Full Case Study: Understanding the Situation*

(2.2.4) Complete reading the System Architecture section (B.6 through B.10). Revise your four diagrams to incorporate the new information.

(2.2.5) Based on your revised diagrams, answer these questions:

(2.2.5.1) Which perspective is the most helpful to you? Of your diagrams showing the four main types of logical architectures: user-oriented, function-oriented, geographically-oriented and device-oriented, which helped you most to understand the situation and to plan the performance test?

(2.2.5.2) How sensitive is your answer to the assumptions you made earlier? If your assumptions varied, for example, how would that affect your diagrams? Should you change your diagrams and the assumptions behind them, to reflect any new information or shrewd new insights?

(2.2.5.3) At first glance, do you have any hunches about potential vulnerabilities or bottlenecks based on the proposed system architecture?

(2.2.5.4) What aspects of this system -- if any -- look straightforward to test for performance and reliability? Why?

(2.2.5.5) Which aspects look difficult to test? Why?

(2.2.5.6) Of the test suggestions from the technical community (see later), and based on what you know so far, which ones would you tentatively approve and incorporate into the test strategy? Which if any would you deny? Why?

(2.2.6) What technical skills and support do we need to test successfully in this environment?

(2.2.6.1) Within the performance test team?

(2.2.6.2) Externally to the team?

## *Description of the Situation, Section 2.B: The System Architecture*

### B.1 ARCHITECTURE OVERVIEW

This section describes the technical infrastructure in which the live system will operate, and the planned initial system configuration. The system architecture has been derived

from a business model, which is in the form of a set of use cases describing the services that need to be supported by the infrastructure. (You will not need to review the use cases to do the exercises.)

The new system will run on a multi-tier server architecture which will be shared by other applications needed for the book club's business operations, such as the billing system. The system's technical environment includes databases, web sites, wireless capabilities, voice call center facilities and high-volume printers.

You do not need expertise as a system architect, system administrator or network engineer to read and analyze this material, which is organized as follows:

B.1 Architecture Overview
B.1.1 Infrastructure Design Goals and Principles
B.1.2 Logical Vs. Physical Design
B.2 Designing for High Availability
B.2.1 Designed-In Redundancy
B.2.2 Designed-In Scalability
B.2.3 Clustering and Fail-over
B.2.4 Geophraphic Dispersion
B.3 Major Tiers and Work Load Distribution
B.3.1 The Front-End
B.3.2 The Back-End
B.3.3 Load Balancing
B.3.3.1 Network Load Balancing for the Front-End
B.4 The Web Sites
B.4.1 The Primary Web Site
B.4.2 Providing Web Services
B.4.3 Proxy Servers
B.4.4 Web Databases
B.4.5 Location of Web Content Storage
B.4.6 The Secondary Web Site
B.5 The Data Architecture
B.5.1 The Data Content
B.5.2 Data Conversion
B.5.3 Database Size
B.5.4 The Database Servers
B.5.5 Data Distribution and Mirroring
B.6 Networks and Communications
B.6.1 The Network Topology at Headquarters

B.6.2 Network Interface Cards (NICs)
B.6.3 Utilization of Network Technologies
B.6.4 The Fax and E-Mail Servers
B.6.5 The Voice Telephone Servers
B.6.6 The Wireless Routers or Servers
B.6.7 The Remote Location Servers
B.7 Other Subsystem and Component Descriptions
B.7.1 The Application Servers
B.7.2 The Print Servers
B.7.2.1 Print Out-Sourcing
B.7.3 The Support Software
B.8 Security Considerations
B.8.1 IP Addresses
B.8.2 Firewalls
B.9 System Implementation
B.9.1 Re-Use of the Existing Equipment
B.9.2 The System Implementation Strategy
B.9.3 Physical Installation and Set-Up of the Equipment
B.10 Architecture Evaluation
B.10.1 Review History
B.10.2 Likely Performance Vulnerabilities
B.10.3 Possible Bottlenecks
B.10.4 Test Suggestions from the Technical Community
B.10.4.1 Database Performance
B.10.4.2 Web Site Performance
B.10.4.3 Maintainability

### B.1.1 Infrastructure Design Goals and Principles

The system designers' goals are to deliver a system that

a) works (i.e., provides the features that fulfill the business requirements);

b) is right-sized and supports the business in a cost-effective manner;

c) meets or beats competitors in responsiveness and availability;

d) provides flexibility and agility to accommodate change; improves reliability through redundancy, and

e) facilitates troubleshooting by promptly detecting problems as soon as they occur during live operation and by pinpointing their causes.

The architecture is based on these principles:

a) the business drives the services, and the services drive the technology;

b) agility to respond to change is a fundamental business requirement, and the architecture is expected to be in flux on an on-going basis;

c) the system is as platform-independent as possible;

d) the system is loosely coupled, in order to minimize bug propagation (so that a code change on one server, for example, does not necessitate code changes on other servers);

e) interfaces will be mainstream and standards-based (i.e., only reasonably mature, non-proprietary interfaces will be utilized);

f) the system can scale up easily to handle the projected growth for the next 5 years; and

g) it is designed for robustness (e.g., fault-tolerant with fail-over capability, load balancing, redundancy and reserve capacity).

The service-oriented architecture uses Internet protocols to interface among semi-autonomous applications. XML, messaging and the web will be utilized to re-architect existing applications and build new, integrated applications.

### B.1.2 Logical Vs. Physical Design

The system architects have differentiated between logical and physical design. The former is a conceptual diagram which shows the resources provided to the system, such as servers, how they connect together and how they will be utilized. This logical design does not necessarily reflect the exact physical implementation, though it should be possible to map from the logical to the physical and vice versa. By contrast, the physical design shows the actual system components and specifies information like each device's brand and model number, processor speed and type, installed memory size, and hard disk read/write speeds and size.

For example, the architects could recognize the need for a user authentication capability. They represent this need in the logical design by a conceptual box labeled "user authentication server". In the physical design, however, this conceptual server may actually be grouped together with other needs. Jointly these needs are supported by only one shared physical server, or alternatively the one conceptual server might be implemented as an interconnected cluster of several physical servers.

At this stage, the logical system design is complete and stable but the physical design is expected to continue to evolve until the system delivery and beyond. The remainder of this section describes the logical design. Physical design decisions are reflected in this logical design description where (a) they are relevant to understanding the test issues, (b) the physical decisions are known and (c) they appear reasonably solid.

The purpose in incorporating elements of the physical implementation into this logical design is to provide enough detail to adequately plan the performance testing. [For the purpose of this exercise, you can assume that a performance test strategy based on the following description of the logical design will work with any specific physical implementation of that design. This assumption is convenient in a book or a classroom but may be unwarranted in your particular organization.] Three questions that we (the performance testers) will need to consider:

- How much architecture detail do we want or need to know in order to test system performance?

- In which areas do we need more precision, accuracy and confidence in the quality of the information available? In which areas are we indifferent? (In the latter areas, the success of performance testing is only loosely related or unrelated to the information quality.)

- How do we accommodate uncertainty and volatility while planning for the performance test?

    - What is the impact on the test plan of architecture details that are not yet decided and of decisions that may be revised?
    - In which areas of the architecture are lack of details or possible later changes most likely to impact the performance test?
    - In which areas of the architecture are lack of details or possible later changes least likely to impact the performance test?

For the remainder of this Section 2.B, the term "server" will be used to mean a *logical*

server, not necessarily a physical one, unless otherwise stated.

### B.1.3 Design Review and Validation

The scope of the performance testing project does not include a critique by the testers of the performance implications of the system design and infrastructure decisions. (Those fools! We testers could tell the architects a thing or two.)

A group of experienced architects were asked to comment on the system design. Their feedback is summarized later (in section B.10.4).

Users and managers who are familiar with the existing systems (i.e., those being replaced), have provided an evaluation of the current quality of service in Section A.11. Some observers believe this history of problems is irrelevant as the systems are being replaced – the new systems will presumably have a new set of problems. Other observers disagree – they believe the prior history could be a good predictor of future behavior. (Which group of observers do you think are right? Under what circumstances, and why?)

## B.2 DESIGNING FOR HIGH AVAILABILITY

The term *high availability* refers to the ability of a multi-server site to withstand hardware or software outages that occur on the site's individual devices. These outages can be planned or unplanned. An example of a planned outage is taking a server down to perform a software update. While the server is down for the software maintenance, the rest of the site stays online and services users. An example of an unplanned outage is a catastrophic server failure. In this case, the rest of the site should stay online providing services to users because the processes failed over to the remaining servers in the cluster. The architecture is designed to protect the data and keep the site up and running.

### B.2.1 Designed-In Redundancy

The system will use multiple devices such as servers and routers, with each logical device dedicated to providing a specific capability (also called a function, service or tier). Each tier can be changed and scaled independently of the other tiers, making the entire infrastructure more robust and more scalable.

The logical architecture therefore utilizes several dedicated-function servers. To

increase reliability, at least two interchangeable physical devices (usually servers or routers) are assigned to each capability in parallel. In the physical implementation, these numerous logical servers actually may be hosted on a smaller number of large physical servers, without compromising the redundancy and reliability sought in the logical architecture.

### B.2.2 Designed-In Scalability

The architects claim that scalability has been allowed for throughput the system design. For example, tables and databases can expand to many times their current sizes without address and indexing limitations.

Additional devices can be added to live clusters and automatically begin sharing the load without disrupting on-going activity. The bandwidth of links can be increased as necessary, and telecom devices like switches are all upgradeable.

Initially, each server is intended to have a single processor. However, the flexibility has been left in the budget to upgrade a few selected servers to dual-processor machines or to double their memory, if necessary, depending on the results of the performance testing.

### B.2.3 Clustering and Fail-Over

As the system is mission-critical, one design goal is to host the applications on a flexible platform that provides scalability, reliability, and availability. Clustering helps provide a solid infrastructure on which to deploy the applications with confidence, satisfying the customer demands. Clustering provides the means to distribute work loads across multiple servers with load balancing, and fail-over capability for application software processing (such as order entry), networks and databases.

The servers in the cluster are intended to fail-over, that is, to back each other up as necessary without a loss of functionality. In the event that one server is disabled, the overall system should remain operational, even if it provides a degraded, slower level of performance to the users until full capability can be restored.

### B.2.4 Geographic Dispersion

While most of the infrastructure will be physically located at the headquarters, there are also two other sites. One is a remote satellite office in the Los Angeles metropolitan area, where about 10% of the staff works. These people need to be connected with their

peers at headquarters, which is some miles away.

The other remote site is a second, work-sharing web site in New York. This East Coast location was selected because (1) it provides an alternative access to the Internet backbone, thus increasing reliability, and (2) it will improve performance for East Coat and European customers. The web traffic to and from external customers and suppliers is intended to be split approximately equally between these two East and West Coast web sites, while the Los Angeles locations will handle all the internal staff demands (on the Intranet).

### B.2.5 Locations and Assignments of the Servers

Following the architecture principles expressed earlier, each logical server will be dedicated to a specialized use and coupled to another redundant, load-sharing server that provides the same functions and services.

The book club anticipates that several servers will be deployed on the West Cost. The logical server configuration at the Los Angeles headquarters site will include dual servers for each function or service, such as the database service and the printing service.

Another pair of servers will be located at the remote satellite office, which is also on the West Coast, to mange the local operations of that office.

The East Coast architecture is a scaled-down version of the West Coast headquarters' architecture, with the major exceptions that there is no need for a LAN on the East Coast, and there is no remote office on the East Coast.

This configuration may change as the system design is refined, or to balance and tune the system, or when the system load changes.

## B.3 MAJOR TIERS AND WORK LOAD DISTRIBUTION

The site's multi-tiered architecture is physically divided into two main tiers, the front-end and the back-end. The clustered front-end provides the core web services such as Microsoft's Internet Information Services (IIS). The clustered back-end provides application and database services.

### B.3.1 The Front-End

We saw earlier that the site plans to use multiple servers to provide Internet access and respond to requests from users. These front-end servers provide web services (using IIS), serve HTML, XML and ASP pages, execute objects called from ASP pages, and so forth.

The front-end tier of the site includes a primary domain controller (PDC), security services, a proxy service, a domain name service (DNS), e-mail and fax services, and a web content staging service. Each of these services is supported by its own logical server.

The PDC is a server that maintains a read-write directory of user accounts and security information. The PDC authenticates usernames and passwords when members log into the network. Members only have to log into one domain to access all resources in the network. A backup domain controller (BDC) will be provided so that the primary domain controller is not a potential single point of failure.

The DNS service resolves the private addresses for the site: addresses of the individual servers on the private network are stored in the DNS database.

The staging service provides temporary stages to test new or revised web pages before they are deployed into the live operation.

The front-end also provides security services such as firewalls.

All Internet access to the back-end must pass through the front-end. All services that are not essential for providing web services are turned off on the front-end servers to prevent unnecessary resource usage and to remove possible attack points. For example, because FTP and SMTP services are not provided as part of the service offering for the front-end servers, these services are either turned off or not installed on the front-end servers. Everything in the front-end is located inside the firewalls.

### B.3.2 The Back-End

While all Internet access to the back-end must pass through the front-end, other internally controlled services access the back-end directly. These include the voice telephone service (which manages the external and internal voice phone access, the Intranet (a local area network or LAN), the remote satellite office service (a wide area network or WAN) and the wireless service (which manages the wireless devices), are connected directly to the back-end.

Back-end servers run in a cluster and provide data services (databases and file shares) for the site, plus all other services not provided by the front-end. The back-end cluster is configured in active-to-active mode.

The back-end cluster provides fail-over capability for services running on the cluster. If one of the servers goes down, due to hardware failure, planned maintenance or any other reason, the other servers in the cluster immediately take over the services of the downed server. The failure of a server does not cause failure of the data services or interruption in service. When the downed server is brought back online, it resumes delivering data services.

The data for both the database and the web content is further protected by being stored on a RAID disk array. (RAID means redundant array of independent disks.) In the event that a hard disk fails, the data will continue to be available, and a functioning hard disk can be hot swapped into the array with no interruption in service.

The back-end servers send periodic messages, called *heartbeats*, to each other to detect failed applications or servers. The heartbeats are sent on a dedicated network, using network interface cards (NICs) dedicated to this purpose. In the event that one server detects a heartbeat network communication failure, it requests verification of the cluster state. If another server does not respond, ownership of resources (such as disk drives and Internet Protocol (IP) addresses) is transferred from a failed server to a surviving server. It then restarts the failed server's work load on the surviving server. If an individual application fails (but the server does not), a server cluster will typically try to restart the application on the same server. If that fails, it moves the application's resources and restarts it on the other server.

### B.3.4 Load Balancing

Load balancing software (or possibly hardware) will automatically share and balance the load among the servers. At the headquarters location it will route transactions between the application servers, route queries between the database servers, and route print requests between the print servers in order to optimize performance. It is anticipated that no extra add-on load balancing tool will be utilized; the balancing will be done by the built-in capabilities of the server operating system, though this decision has not become final yet. If however this load balancing capability servers is provided by hardware, a separate device is expected to be physically located upstream in the work flow, prior to the devices being balanced.

#### B.3.4.1 Network Load Balancing for the Front-End

With network load balancing (NLB), the web servers work together in a cluster to handle the web traffic. NLB is configured on each server in the cluster to respond to the same virtual IP address and domain name. The scalability and load balancing occurs by the primary domain controller directing resource requests across the front-end servers to best balance the load for the site. The load-balancing algorithm determines which server actually responds to a user request.

When the traffic increases beyond the capacity of the site, a new front-end server can be configured with the NLB settings. When the new front-end server is booted up on the network, it will dynamically join the existing NLB cluster and immediately begin sharing the load with the other front-end servers.

When the NLB detects a server that is not responding to network requests, it removes it from the cluster. The remaining nodes pick up the load of the server that is down, to keep the site alive.

## B.4 THE WEB SITES

### B.4.1 The Primary Web Site

At the primary site in the Los Angeles headquarters, the web servers will be connected to routers, which in turn connect to the Internet through an OC-48 high-capacity data link with the bandwidth of 2.4 gigabits per second. This is sufficient capacity to accommodate up to 300 Web sessions (assuming each requires 1 megabit per second or mbps bandwidth), 300 simultaneous telephone calls (each requiring 128 kbps bandwidth for VoIP technology), and 300 Web dial-up sessions using 56 kbps modems, assuming no data compression technology is used, with an additional unused capacity remaining of 2 gbps for peak surges in demand and for internal data transfers, e.g., for data mirroring. Though only one OC-48 data link is leased, the telecom carrier automatically provides a built-in hot back-up for this link.

The book club expects that the web servers will be accessed from a variety of browsers on a variety of remote platforms.

There will be firewalls and other security functions on the web servers which may slow their performance. (See the later section B.9, entitled: “Security Considerations”.)

### B.4.2 Providing Web Services

The front-end web servers deliver the same web content and share the work load,

responding to HTTP requests and distributing web content. User requests are made using a URL that all of the front-end servers can respond to. The front-end servers access the Web site content data located on the back-end cluster file share service.

All objects necessary to provide Web services are installed and registered on each front-end server. These include objects that are called from ASP pages. ASP pages for the site can either be loaded on the front-end servers' local disks, or kept on the back-end cluster file share service.

### B.4.3 Proxy Servers

These servers sit between the client applications, such as the web browser, and the real web servers. A proxy server intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server. A proxy server can dramatically improve performance for groups of users, because it saves the results of all requests for a certain amount of time. To improve security, the proxy servers will also be used to filter requests.

### B.4.4 Web Databases

The web servers will have their own databases, physically located on these servers, so they are not totally dependent on the database servers. These web databases will contain redundant copies of the same data and will be used to store frequently downloaded data, such as FAQs, and to help isolate the web site visitors from the system users inside the book club.

The web server databases will not contain all the data needed to answer queries or process orders, so these servers will need to access the database servers for some data. Although the web servers are physically connected to the database servers, some of the confidential company data on the internal databases is intended to be essentially invisible to web visitors for security reasons -- in other words, this data cannot be accessed by the web servers.

### B.4.5 Location of Web Content Storage

Web site content (HTML, ASP pages, and so forth) is stored on the back-end clusters instead of local disks on each of the front-end servers. This is done because:

- Using a RAID disk array makes the data more available.
- In the event of a server cluster node failure, the file share service can fail-over to a

remaining server.

- It is easier to manage site content and keep it synchronized when it is located in one place rather than distributed across the local disks on each front-end server.

### B.4.6 The Secondary Web Site

The secondary web site in New York will have six logical servers, with two dedicated to servicing the web traffic, two to the databases and two for applications processing. These will mirror their Los Angeles counterparts as appropriate for the functions they support, and will be connected on their own LAN.

## B.5 THE DATA ARCHITECTURE

The database services are provided by Oracle database software running on the server cluster in active-to-active mode, meaning that dual database servers provide services rather than having one server to provide all the services and the other wait on hot standby (active-to-passive mode).

### B.5.1 The Data Content

The main types of data stored and used by the system are:

- Orders (tracks orders from customers; billing)
- Catalog (all the books the club offers)
- Inventory (all the products on hand)
- Customers (all the customers and customer profiles; members are simply customers who choose to enroll in the club)
- Vendors including vendor orders (all the orders the club makes to its suppliers)

### B.5.2 Data Conversion

A new database is currently being converted from other existing databases of book club members, orders and books. The senior managers have identified the database conversion as a complex, high-risk activity, and they are concerned that after the

conversion several cycles of database tuning may be needed before the system performance goals are met.

### B.5.3 Database Size

When it has been converted from existing files, the database will contain approximately 100,000 active members. (Active members are those who have notified the club of their desire to receive e-mails with promotional announcements or monthly catalogs of books, or who have ordered a book within the last year. The typical member's levels of activity, e.g., orders per month, are provided later in Section 2.E.) The database will also contain another 200,000 inactive people who have ordered at least one book in the past, but not recently, or who have expressed an interest in the book club. The data on these other, inactive people is used in marketing campaigns. People are deleted from the database after three years with no activity.

### B.5.4 The Database Servers

While the technology exists to support real-time concurrent updates to multiple database servers, the designers have decided that its costs and risks do not justify its use except in the most critical business transactions. The database servers will use RAID (redundant disk) technology, so that each database server in itself has redundant copies of the data.

A complete back-up copy of the database will be taken once every night. During this back-up process, which is expected to be completed within a period of 30 minutes, user response times may be slow. Few users are expected at the time, probably 3.00 am, so that slow responses are a minor issue. No guarantees have been made to the user community as to the worst-case performance of the system in this situation. Nevertheless, since the managers want to know if the user response times are likely to be extremely slow during the back-up, measuring those response times is within the scope of this project.

### B.5.5 Data Distribution and Mirroring

Each of the database servers will contain a partial copy of the full database, so within the duplicated portion the same data is stored on the other database servers. Data updates will be made routinely (but not necessarily concurrently or immediately) to all copies of the database, and these copies will be periodically and automatically monitored for consistency by the database management system (DBMS). This monitoring in itself requires significant message traffic among the database sites.

The mirrored or duplicated data includes the full product (book) catalog but not all products' current inventory levels, the full set of members' basic data (account number, name, etc.), and their currently active orders (i.e., those about to be picked and shipped).

The objective of mirroring is to improve data accessibility, back-up and load distribution, by providing more than one source of data, which in turn should improve performance. Though there are trade-offs. Employing a single-site integrated database instead of distributed ones does have the advantage of easier centralized control. In a decentralized environment, the proliferation of distributed databases can mean a lack of consistent, timely, consolidated information at the corporate level. However, the reliance on one site compromises accessibility, performance, and local control over information. Thus the DBAs have made the decision to distribute and mirror the data. They have documented the specifics of how the data will be distributed and accessed in a thick and not especially user friendly document. The summary provided here will just have to be enough to plan the performance test.

Data storage for the web site is managed by two servers in a cluster connected to a shared RAID disk array. The server cluster provides availability in the event of a server failure, and the RAID array provides availability in the event of a disk failure. The disk technology provided can detect potential disk failures before they happen. If a disk failure is predicted by the system, the failing disk can be hot swapped out of the RAID array and replaced with no loss of service. RAID arrays can be implemented in software or hardware for increased data access performance.

## B.6 NETWORKS AND COMMUNICATIONS

### B.6.1 The Network Topology at Headquarters

While book club members and visitors can access the system only through the web site, the internal part of system used by the book club employees will operate on an Intranet (a private, internal client/server network). The Los Angles area will have approximately 250 users on the Intranet, with 225 in fixed locations with one workstation per user, and 25 mobile wireless users with hand-held devices.

Of the 225 fixed-location workstations, 200 will be physically located at the book club headquarters, which is an office building in Los Angeles, and will be on a local area network (LAN). The other 25 fixed-location workstations will be located at a satellite

office, which is several miles away and which will be connected to the headquarters building by a wide area network (WAN). The 25 wireless devices will be used in the warehouse, which is attached to the Los Angeles headquarters building.

### B.6.2 Network Interface Cards (NICs)

Each server has two 100-Mbps Ethernet network interface cards (NICs). The TCP/IP protocol is used throughout the site.

In the back-end servers running in a cluster, one NIC is connected to a private network providing access to the other servers through a 100-Mbps switch. The other NIC provides the cluster heartbeat mechanism and is connected to the other cluster server by an Ethernet cross-over cable. In the front-end servers providing services, one NIC is connected to a 100-Mbps switch that is connected to a network that routes to the Internet.

### B.6.3 Utilization of Network Technologies

Within the headquarters building, the fixed-location clients and servers will be connected by a fast Ethernet local area network (LAN), rated at 100 Mbps. If the performance measurements indicate that the LAN is a major bottleneck, the speed of the LAN could possibly be increased to a gigabit per second (using gigabit Ethernet). However, an allowance for this contingency has not been included in the project budget.

The web and CTI (telephone) servers will be connected to the OC-48 data link, which can be shared by both the web and voice telephone traffic. As mentioned earlier, this link has a rated capacity of 2.4 gigabits per second. The overhead for protocol conversions from external networks, such as SONET to internal Ethernet LAN, are assumed not to be a significant issue and will not be measured as part of this project.

Internally within the secondary web site and also within the L.A. satellite office building, the clients and servers will be connected by traditional Ethernet LANs, rated at 100 Mbps.

The wide area network (WAN) connecting the headquarters and the satellite office will utilize dedicated T1 lines.

### B.6.4 The E-Mail and Fax Servers

These provide e-mail and fax services to the organization’s staff members.

### B.6.5 The Voice Telephone Servers

The voice telephone servers will be used to coordinate incoming calls with customers' profiles, so that the person answering a call has ready access to the caller's account status. For example, a customer's data will be retrieved automatically, based on the telephone number of an incoming call, and displayed on the user's screen as he or she answers the call. The voice telephone service will be supported by call center software which resides on the voice servers.

Voice traffic will use voice-over-IP (VoIP) technology, where conversations are communicated through digitized streams of TCP/IP data packets.

The voice service, for both incoming and outgoing calls, will be handled by the same OC-48 data link that handles the web service.

### B.6.6 The Wireless Routers or Servers

The wireless routers or servers will be used to communicate with the hand-held devices in the warehouse. These devices will help direct the order fulfillment, specifically the picking, packing and shipping of books.

In the future, the hand-held devices will be shared with other application systems in addition to the order processing system. The main other application is expected to be inventory management, where the hand-held devices will be used to take and report inventory counts and report out-of-stock conditions directly from the warehouse floor. A wireless-based inventory management system will be implemented, but not until at least 9 months after the order processing system goes live. The inventory system is expected to account for approximately 33% of the total wireless message traffic. (The order processing system will account for the other 67%.)

The system designers have not worked with wireless technology before.

### B.6.6 The West Coast Remote Location Servers

Two more servers will be situated at the satellite office in the Los Angeles metropolitan area. These servers will support a local area network for the 25 client workstations in the satellite office, and will connect the satellite office to headquarters by a wide area network. The business group which is located at the satellite office, and which is the main group supported by these servers, is the catalog publishing group. The role of this group is described in other section A.4 of this document, Part 1 of the case study.

## B.7 OTHER SUBSYSTEM AND COMPONENT DESCRIPTIONS

### B.7.1 The Application Servers

A full set of the application systems will reside on each of the two application servers, so that both servers will have the capability to support all the system features and process all transactions. The set of applications includes the billing, inventory management and purchasing systems (purchasing books from publishers and distributors).

### B.7.2 The Print Servers

The print servers are connected to eight high speed laser printers, which will be shared by several users, and which are located in the work areas close to their main intended users. These printers are used to print in high volume the paper copies of the book club catalog. Although the catalog is available for on-line browsing, printed copies are available on request. Approximately 5,000 paper copies of the 300-page catalog are printed and distributed per month. The printers are also used for a variety of internal business reports.

In addition, dedicated local printers will be directly connected to individual personal computers and normally will not be shared across the network. Measuring the print times on these local printers has been declared to be outside the scope of this test project. The impact of the localized printing on overall network performance is believed to be minimal.

#### B.7.2.1 Print Out-Sourcing

The book club managers have debated out-sourcing the catalog printing, with no consensus likely anytime soon. One group argues that a specialized print contractor would be faster and cheaper. Another group argues that for the foreseeable future the catalogs are critical to the business and thus cannot be outsourced.

If the printing is outsourced, the master copy of the catalog could be delivered to the vendor electronically or by courier service in the form of a hard disk.

### B.7.3 The Support Software

The decision has been made to use Windows XP on both the servers and on the clients.

The database management software used by the book club will be Oracle 9i.

The web servers will run Apache software to service Internet traffic.

The routers will run Cisco's IOS operating system.

The e-mail services will use Microsoft's Internet Explorer browser and Outlook e-mail software.

## B.8 SECURITY CONSIDERATIONS

### B.8.1 IP Addresses

For security reasons, the servers all have two Ethernet adapters, each with different IP addressing. The servers communicate with each other on a private network, and only the front-end servers have IP addresses that are publicly accessible. To prevent malicious attacks, this architecture prevents direct access from the public network to the servers containing site data. It is possible to have just one Ethernet adapter in all of the front-end servers, and provide connectivity to the back-end servers, if the servers are all configured with publicly-accessible IP addresses. However, this would expose the site data (on the back-end servers) to attacks from the public network. DNS is used for all name resolution and there is a DNS server running in the site specifically to handle name resolution for the privately addressed interfaces. The only publicly accessible IP address on any of the servers in this site is the virtual IP address that the front-end servers respond to.

### B.8.2 Firewalls

A firewall can be implemented as software or hardware, the latter in the form of a self-contained unit.  are relatively inexpensive and ubiquitous, and thus can and should be placed at several points in the network topology. The decisions on firewall placement are not final yet. Possibly the web sites will be outside the firewall(s) and the web services behind them. The web services utilize the databases, which are definitely behind the firewalls.

### B.8.3 Trade-offs of Security and Performance

Security controls like encryption are expensive, as they degrade performance, require more resources to run, or both. Security controls are implemented in the parts of the

application that asks for user passwords and credit cards, but are not necessary for a catalog page look-up. The security policy guiding the ordering system is as follows: only user-sensitive financial data is protected.

## B.9 SCALABILITY CONSIDERATIONS

### B.9.1 Application Processing Scalability

The application software developers say that they have written scalability capabilities into their software. They have attempted to exploit major two trends in the advancement of processor hardware: (1) the move to 64-bit computing, and (2) the move to highly parallel in-chip processing. Future chip throughput increases are expected to come not so much from faster clock rates, which have been the primary source of semiconductor speed increases over the last 30 years, but instead from increasing the parallel activities on a chip. Application software must be designed specifically to take advantage of 64-bit computing and parallel hardware processing -- the scalability does not happen automatically. (As an aside: how would you test this scalability works?)

### B.9.2 Database Scalability

Xxx

### B.9.3 Network Scalability

xxx

## B.10 SYSTEM IMPLEMENTATION

### B.10.1 Re-Use of the Existing Equipment

The parts of existing infrastructure which can be re-used (equipment and facilities which are already in place and support the existing applications), will be integrated into this new environment. The equipment which will be carried over to the new operating environment has already been included and counted in the inventory of equipment for the new system.

The parts of the existing infrastructure which cannot be re-used, such as older clients and servers, will be discarded after the new environment is operational.

There are no hand-held devices or wireless systems currently being used by the book club.

### B.10.2 The System Implementation Strategy

After considerable debate, the managers have decided not to migrate to the new order processing system in a gradual series of phases. Instead, they plan a “flash cut” one-time transition from the existing systems to the new one: the old systems will be switched off, data transferred and the new system switched on within a period of minutes. This avoids the need to run mixes of the old and new systems in parallel during a cut-over period.

The implementation team plans one or more trial runs of the conversion in order to mitigate the risks, and is developing a roll-back contingency plan in case the transition does not work.

### B.10.3 Physical Installation and Set-Up of the Equipment

Our performance test team is responsible for installing, configuring and checking all equipment used in the test lab.

Our performance test team should not have to test that equipment has been correctly installed and configured in the live environment, even if we choose to do some testing in this live environment. (For example, we could test after this equipment has been installed but before the new system is enabled.)

An experienced contractor will install, wire, set up and configure the computer and network equipment needed for the new system. This contractor will test and confirm that this equipment has been installed and works, and also will confirm that the support software, such as Windows XP and device drivers, has been installed correctly.

## B.11 ARCHITECTURE EVALUATION

### B.11.1 Review History

A group of experienced architects has reviewed the proposed system architecture and concluded that it should be able to support the system’s functional requirements. They also believe that the equipment ordered for the site is probably sufficient to handle the work load, though this is based more on a high-level comparison with other similar sites rather than an in-depth analysis.

### B.11.2 Likely Performance Vulnerabilities

The managers of the book club business groups have compiled this list of the perceived threats to the new system performing adequately.

The likely sources of spikes in demand on the system from external users (non-employees) are responses to promotions in the first hours after they are broadcast, requests to download the e-catalogue in the first hours after it becomes available, external events outside the book club's control like Oprah blessing a book, and denial of service attacks. Heavy internal demand (caused by book club employees) and thus contention for resources are likely to occur when printing the paper version of the catalogue to mail to customers.

Uses which are resource-intensive, e.g., likely to consume a high level of system resources per event include book searches, data mining (this not part of the order entry system, but it runs on the same shared servers), downloading graphics and video clips of authors and books, the ad hoc management reporting and the database backup (the nightly "crawl").

System uses which are particularly timing sensitive include the call center phone calls, the database access for phone call support, and senior management and VIP queries (especially high priority ones).

Other background applications share the same infrastructure and may interfere or compete significantly for system resources. The biggest sources of contention are likely to be the billing system and spikes in e-mail traffic. Since these business managers have no knowledge of the system internals, vulnerabilities caused by inadvertently designed-in bottlenecks or poor implementation (e.g., inefficiently written software code) are not addressed in this section. (See the architecture evaluation performed by a review team of experienced architects, later in the Description of the Situation, Section 2.B.)

### B.11.3 Possible Bottlenecks

The architects have identified some possible bottlenecks, based on their review of the system design on paper. None of these possibilities may actually become bottlenecks, and eliminating them will incur high additional costs for hardware, networks, data storage, software and support. The executives want to right-size, not over-engineer, and are not convinced these costs are necessary. They have decided to wait until the performance test confirms the problems are real before spending more money on

capacity. The possible bottlenecks are:

a) The distributed database, which is shared across multiple applications in a data-rich, highly data-dependent environment.

b) The nightly database back-up, when other data-related transactions may slow to a crawl.

c) Shared servers across multiple applications. For example, during a bill processing cycle the billing system may pre-empt the ordering system on the application servers.

d) The printers, which may not be hard-wired directly to the print servers, but on the local area network (LAN). The reviewers had a split opinion about the printers. Some believe that the monthly catalog printing will be a bottleneck. Others disagree, pointing out that the electronic print file has to be downloaded to the printers only once, regardless of how many thousands of copies will be printed.

e) The voice or computer-telephony interface (CTI) servers also may be on the LAN, leading to the possibility of heavy telephone traffic interfering with other work and vice versa.

f) The web site access and the voice call center apparently share the same telecommunications lines.

g) Security functions, such as firewalls, digital certificates, and encryption of sensitive data (e.g., credit card transactions), are likely to be resource-intensive but apparently are not supported by a dedicated server. Separate front-end proxy servers, firewalls, e-mail software, etc.; reduce the load on the server.

Every system contains bottlenecks. This number of possibilities (eight are listed above) is not unusual, and is not an indicator of an incompetent design. No opinion or information on this topic of possible bottlenecks is available from the system architects and developers, other than they are confident their design will work.

### B.11.4 Test Suggestions from the Technical Community

The technical professionals (system designers, DBAs, web site administrators, etc.) have suggested that the performance test team tackle the following questions. (Caution: some of the requestors may not know as much about performance or testing as they

think.)

*B.10.4.1 Database Performance*

In database test mode, have individual tables and files been allocated sufficient disk storage to accommodate the anticipated volume of production data?

On the web servers, do active temporary data areas (such as cookie and shopping cart storage) become highly fragmented over short periods of time?

If data is not being replicated, then is the DBMS able to successfully roll back a transaction that spans several servers when one of the servers is unable to commit its portion of the transaction?

Splitting the data among distributed databases is likely to generate more network traffic. Will splitting the data worsen the performance of the web site due to increased network congestion?

*B.10.4.2 Web Site Performance*

Does the load balancer spread the work load effectively and not compromise the integrity of the web site's functionality? (This is especially important for transactions that require the user to enter data on more than one web page in order to complete a transaction.)

Are web visitors being correctly re-routed to the most appropriate mirror?

What happens if the primary site goes down?

*B.10.4.3 Maintainability*

Can one or more of the servers in any functional tier be "hot swapped", or additional servers added, without having to temporarily take the site off-line?

Can a new version of system software or application software be implemented without taking the site off-line? (Implementing standard housekeeping procedures on a 24x7 web system is more challenging and potentially more prone to error than a traditional client/server or mainframe system.)

## *Exercise 2.3: Specifying the Performance Requirements*

### Introduction

Testers commonly face untestable requirements {e.g., vague and unspecific, ambiguous, not measurable, etc.). They may have no choice but to refine or re-work them so the requirements are usable. This can be a time-consuming, politically charged and thankless task.

The purpose of this exercise is to develop a workable set of performance requirements by refining the set provided earlier (see section 2.A, part A.10). These existing requirements have been deemed inadequate. For brevity, develop only a subset of requirements, numbering three testable requirements in all.

### Instructions

First, re-visit and review again the business objectives from part A.8 and performance goals from A.10.

Second, answer these questions:

(2.3.1) Where do we need to set goals? Performance goals could be set for a very large of situations. Which 5% of the performance envelope is most worth testing, monitoring and caring about? Put another way, which 5% of the system behaviors are in areas where the performance requirements are worth defining? It helps to address the following subsidiary questions as stepping stones to deciding where to set the goals:

(2.3.1.1) Which users have priority, in terms of receiving the best performance?

(2.3.1.2) Which types of work have priority?

(2.3.1.3) Which system uses are likely to cause the most user dissatisfaction or business losses if performance is poor?

(2.3.2) How aggressive do the service levels need to be, i.e., the goals that are user-visible? (Service levels are goals that are meaningful to users. Internal disk utilization, for example, is not user-visible.) It helps to address these questions:

(2.3.2.1) Who are our main competitors? How do customers differentiate among us and the remainder of the competition?

(2.3.2.2) How do customers and potential customers evaluate the quality of service of on-line book clubs, and other competitors (e.g., retailers and wholesalers)?

(2.3.2.3) What are the measures or metrics by which the book club is compared with other book providers -- and thus competes for customers?

(2.3.2.4) In comparative evaluation with competitors, in which performance-related areas do we (a) need to beat the competitors, (b) meet them or (c) do not care as performance is irrelevant? (This overlaps the prior questions to some extent.)

(2.3.2.5) If we are not in a competitive situation, what stated goals and unstated expectations are there for user productivity?

(2.3.3) Under what demands do these service levels need to be met?

(2.3.3.1) Under what expected load volumes do they need to be met?

(2.3.3.2) With what other concurrent demands (overheads and background noise) that are imposed on the shared resources?

(2.3.3.3) What rates of growth does the system need to accommodate?

(2.3.4) With what resources (and therefore, at what cost) must the service levels be met?

(2.3.4.1) What types of resources does the system need to run on? (Do not bother to list these if they are obvious.)

(2.3.4.2) What quantities of these resources are normally available to the system?

(2.3.4.3) If you can identify them, which resources do you believe are the most important or critical to the success of the operations?

- Under typical load?
- Under peak load?

(2.3.4.4) What are the desired norms for resource utilization, for each critical resource?

- Under typical load?

- Under peak load?

(2.3.5) What are the desired norms for reserve or spare capacity, for each critical resource?

- Under typical load?
- Under peak load?

(2.3.xx) Are there any exceptions to the performance goals? For example, should the normal response time expectations be ignored during the period of the nightly database back-up?

(2.3.6) Screen the original set of performance goals and discard any irrelevant ones.

(2.3.1) Re-write each of the remaining original goals so it is significant, focused, correct and testable.

(2.3.2) Review your revised set of goals. Are there important aspects that are not adequately covered, so that additional goals should be added? What are these new goals?

Third, document three significant performance requirements. See Appendix G for a suggested format for your answers.

We will visit some of these issues again in later exercises as we refine our answers, specifically 2.8 (calculating the load) and 2.12 (developing test scenarios). This overlap means there is partial redundancy, which is deliberate in order to examine the issues from more than perspective.

## *Exercise 2.4: Performing the Initial Impact Assessment*

### Introduction

The purpose of this exercise is to perform an initial assessment (IIA) of the significance of the likely performance issues, and decide whether a formal performance project is justified.

To make the exercise manageable, we will assess only one part of the system – the local area network (LAN) which supports the Intranet at the Los Angeles headquarters. Normally, on the job we'd first ascertain whether the LAN is the best place to begin. We want to begin with the system component that is the easiest to analyze and also is likely

to be a bottleneck. Usually impact assessments have a broader scope and examine all areas where the impact is likely to be non-trivial, not just one area such as the LAN. So we'd also probably look at more than one area on the job, i.e., we might analyze the web services and the database as well as the LAN.

Section 2.E provides the volumes of LAN traffic, and the transaction lengths for that traffic.

**Instructions**

(2.4.1) Estimate the utilization of the LAN during for the average minute and for the busiest minute of a peak hour.

- Estimate the LAN utilization in terms of bandwidth, i.e., by the percentage of the theoretical maximum capacity that is used.

- Is the new system demand likely to have a material impact on the LAN? If so, the organization should upgrade the LAN, or undertake a performance test or a performance improvement project.

- Limit your analysis to the LAN for this exercise (to make the exercise manageable within the time available).

- Review the simplifying assumptions below. These are intended to expedite your calculations without introducing too much inaccuracy. Note that you are not obliged to use these assumptions. Feel free to replace them with your own assumptions.

- Any there any assumptions below that you question or feel uncomfortable with?

(2.4.2) For this exercise (which is a quick rough assessment), what transaction lengths should we use?

- Should we use the average transaction lengths or the medians?

- When we proceed later to actually testing the LAN, should we continue to use fixed-length transactions provided that their length is the same as the average (or median) in live operation?

- Or should we test with a representative variety of transaction lengths, using a mix

with a profile that seems to match real life?

(2.4.3) Estimate the utilization of the LAN during periods of peak load (specifically during the typical busiest minute of a peak hour). Determine which of these four scenarios causes the largest load on the LAN under peak load:

(a) Announcement of a new e-catalog and the resulting orders.

(b) Monthly catalog printing.

(c) Database back-up.

(d) Database mining.

(2.4.4) What assumptions did you make, either beyond or in contradiction to the simplifying assumptions below?

(2.4.5) Based on your analysis of the system's impact at the LAN at the Los Angeles headquarters, decide whether you should advise the senior managers to fund a performance test for the new system.

(2.4.6) The architects and the design reviewers have debated whether fast Ethernet is really needed, or traditional Ethernet will suffice. The former has a rated capacity of 100 Mbps, while the latter is rated at 10 Mbps and is cheaper. What is your opinion in this debate, based on your IIA?

(2.4.7) Was it shrewd or foolish to focus this assessment on the LAN rather than some other aspect of the integrated system, such as the web services or the database?

(2.4.7.1) Relatively, how easy is to analyze the LAN? How well do we understand the LAN versus other aspects such as the web services, in terms of (a) how the LAN is being used, (b) the underlying technology (how an Ethernet LAN works) and (c) the common issues encountered by users (LAN administrators)?

(2.4.7.2) How likely is it that the LAN will be a bottleneck or at the center of critical performance issues for this system?

(2.4.7.3) How comfortable are we that we have developed a realistic model of the LAN, with good predictability?

(2.4.7.4) How much did the simplifying assumptions compromise the outcome of the IIA? In other words, what is the risk that these assumptions have led us to the wrong conclusion?

(2.4.7.5) This question is optional – answer it only if you are knowledgeable about LAN technology and operations. I gave you an apparently complex situation here and then supplied assumptions that make the assessment exercise relatively straightforward. Is there a better way to simplify the situation? Can you recommend a better LAN model, i.e., a better set of simplifying assumptions? To paraphrase Einstein, a model should be as simple as possible but no more. If there is a better way, how do you recommend we model (simplify) the situation?

(2.4.7.6) Are we being realistic? Is an upgrade of the LAN so straightforward, inexpensive and quick that no impact assessment is needed? If so, is it is more effective to simply skip this IIA and invest the avoided costs (from bypassing the IIA) to blindly upgrade the LAN? This exercise revolves around the LAN: you were asked to assess the need for performance testing based on your estimate of the load impact on the LAN. A LAN expert might immediately conclude – without doing an impact analysis – that actually there is no need for performance testing. This is because, the expert claims, LANs are so cheap to upgrade and so high in bandwidth that even if there was a bandwidth problem, the testing process would be far more expensive than simply fixing the problem after it becomes manifest. Do you agree with the expert?

This set of questions (2.4.7) raises an important point about the effects of content. I use exercises in this book that don't require deep knowledge of specific technologies, so you and other readers can do them successfully. However, I don't know of any way to develop content-free exercises. People who have the relevant knowledge will apply it, while people who don't have any knowledge may be intimidated and confused. I have not included tutorials on the underlying technologies, since good tutorials are widely available on the Internet. Go there first if you are unfamiliar with the technologies.

**Simplifying Assumptions**

We need to make some simplifying assumptions in order to perform a quick assessment. In this situation, the suggested assumptions are:

## *II. The Full Case Study: Understanding the Situation*

- For the moment, we will assume we can ignore the question of whether the new system is able to meet its response time requirements (such as responding to internal workstation queries within 2 seconds, 90% of the time or more), and focus on the transaction throughput only.

- We will consider only the capacity of the LAN to handle the throughput, not the capacity of other parts of the system such as the database servers and the web servers.

    - Do we have to assume here that the LAN will reach its maximum capacity before other links or components such as the servers reach theirs? In other words, if a bottleneck occurs as the load on the system grows, do we need to assume that the limitation will happen first in the LAN? Under what circumstances would this assumption be necessary?

    - Are there variations of this exercise in which we do *not* have to assume the LAN is the limiting factor if we intend to analyze the capacity of the LAN only and no other parts of the system?

- Each interaction with the new ordering system normally includes both an input and a response. Since we expect an input to be followed fairly rapidly by a response, we can calculate the total traffic load on the network for each transaction by simply adding together the lengths of the input message and the response message for that transaction, as follows:

| Type of Transaction | Average Length per Transaction (KB) | | | Median Length per Transaction (KB) | | |
|---|---|---|---|---|---|---|
| | Input | Response | Total | Input | Response | Total |
| Book Search | 0.1 | 16 | 16 | 0.1 | 32 | 32 |
| Order Change | 2 | 2 | 4 | 2 | 2 | 4 |

- It is more appropriate to use the average lengths and not the medians. (See Appendix A for an explanation of mean or average, median and mode.)

- How dangerous is it to use a fixed length, regardless of whether that length is the mean, median or mode? Is the earlier question about using the mean, median or mode missing the point? Is the only credible way to model the LAN's behavior to simulate a representative mix of transactions of varying lengths, based on the operational profile? Would the use of varying lengths produce an answer that is

sufficiently accurate to justify the extra effort?

- Can we safely simplify the problem by ignoring 90% of the transaction types carried on the LAN?
    - The LAN carries various types of transactions, such as book search requests and results of order status queries, as part of the order entry and fulfillment.
    - Of these types of transactions, the most popular 10%, in terms of their contribution to the total load on the LAN, account for 90% of the bits carried, under both average and peak conditions. (By contrast, most of the order entry and fulfillment transaction types – the other 90% -- are infrequently used, and account for only 10% of the bit traffic for the ordering system.)
    - In other words, if we calculate the load contribution for each transaction by multiplying its average length by the expected volume of that transaction occurring within a time interval, the top 10% will predominate.
    - The accuracy required in the load calculations for the initial impact assessment does not have to be better than within plus or minus 15%.
    - If we ignore the bottom 90% of the types of system transactions in this IIA. Will or answer still be within the allowable margin of error?
    - Better yet, can we merely add another 10% to the load calculated from the predominant 10% of the ordering system transactions, and be confident we have a highly accurate answer?
- The LAN also carries traffic that is not part of the ordering system. Is the impact of the other systems which share the same infrastructure and also use the LAN, especially the billing system, likely to be significant and thus cannot be ignored in the IIA?
    - What data do you need to answer this question?
    - And what assumptions are you likely to have to make?
- Do the following assumptions help or hinder the analysis, or re they irrelevant? Is

each one reasonable?

- Under normal working conditions, the load on the LAN should not exceed 40% of the theoretical maximum capacity of the LAN, as measured over a one-minute interval.

- Under peak demand conditions, the load on the LAN should not exceed 85% of the theoretical maximum capacity of the LAN, as measured over a one-minute interval.

- The demand in the peak minute of an hour usually is 10% of the total demand for that hour.

# III. DETERMINING THE PERFORMANCE TEST APPROACH

### Introduction

Now that we have become familiar with the background and context, we need to identify our overall approach to testing the system. We need to address whether to outsource the testing, select the most appropriate test methods, establish the test focus and coverage, and prepare the test work loads and the test environment. The next several exercises examine how to perform these ctivities.

## *Exercise 2.5: Deciding Whether to Outsource*

### Instructions

Review the following lists of the advantages and disadvantages of outsourcing the performance testing effort.

Decide whether, all in all, it is better to outsource or to test in-house, through these steps:

(2.5.1) Review the following lists of outsourcing advantages and disadvantages, and the outsourcing work sheets.

(2.5.2) Customize the outsourcing work sheets for this situation, by deleting any points which are irrelevant and by adding any new relevant points which have not been listed.

(2.5.3) Weight each remaining point with a score based on its relative importance in this cultural and technical environment, on a scale from 1 (minor significance) to 3 (critical).

(2.5.4) Assign a second score to each point on the lists, which represents the degree to which the factor is an issue on this project, on a scale from 1 (minor significance) to 3 (critical).

(2.5.5) Multiply the first and second scores together for each point, and sum the totals of the scores for (a) advantages and (b) disadvantages.

(2.5.6) Am I advocating an invalid use of mathematics here: multiplying subjective factors together?

- Is multiplication the right formula? Should we add the numbers instead? What formula if any would you use?
- Is the multiplication of numbers on ordinal scales mathematically meaningless?
- Even if meaningful, does multiplication bias against high magnitudes?
- Does this alternative work -- instead of calculating the magnitude, just select it subjectively based on the other two numbers.
- Is there an advantage in that alternative approach, in that there is no pretense to scientific rigor?

(2.5.7) Is it necessary or even helpful to fill out a table like the one I propose here? Would you be more effective if we discuss the issues, identify the risks and benefits, then make a decision based on how attractive the benefits are and how well we can manage the risks? (Discussing the issues often is the most important part, but it is missing from my earlier description of the decision process.)

(2.5.8) Decide what conclusions if any you can derive from this quantitative analysis. The results may provide insights into whether to outsource – or may be numerical nonsense.

(2.5.9) Develop a justification for your decision, to either outsource or not.

**Introduction**

Many organizations are contracting out part or all of their testing efforts. This makes sense, if the organization does not have sufficient internal resources or the expertise internally for their testing projects. There are several valid reasons to outsource the testing effort:

Testing is resource intensive, and the demand for testing is cyclical, with peaks occurring when major systems are being implemented. It is more economical to staff on a project-by-project basis than have a "standing army" waiting for the next war or test

(there's not much difference).

Specialized testing expertise (e.g., internationalization/localization testing, performance and stress testing, or knowledge of a particular automated test tool) may be easier to rent than to train and build internally.

Specialized test environments, equipment and facilities may be needed which are not available internally.

External testers also can bring a more objective perspective to a system than the insiders. If the internal team is pressured to meet a delivery deadline, or has lost some of their political and intellectual independence, the presence of the outsiders will help to balance the situation.

In addition, external testing may be required to certify products. VeriTest, for example, has been chosen by Microsoft to certify that third-party products are compatible with Windows. In accounting, a certified audit by definition can only be obtained from an external source.

**ADVANTAGES OF OUTSOURCING**

- May be easier to outsource load testing than feature testing, because the vendor does not need as much domain expertise as for functional testing. (This can be dangerous, however, even in load testing.)
- Lower initial investment in tools and equipment.
- Lower project lead time (this is not guaranteed) -- do not have to learn tools, acquire equipment, etc.
- Lower risk -- with the right vendor -- because of in-depth expertise.
- Another voice of advocacy (the vendor's) to persuade senior managers and users of the importance of load testing.
- Better scalability, because the vendor's test lab has oodles of equipment and virtual user licenses.
- More objectivity and credibility of the test results (maybe this is a bad sign, if outsiders have more credibility than employees).

- If there is bad news to communicate ("it won't work"), the vendor is the messenger.

**DISADVANTAGES OF OUTSOURCING**

- Eventually with re-testing the accumulated vendor costs may be much higher than in-house.
- Never get to build and establish the expertise in-house.
- Sometimes the lack of internal expertise in performance testing means the client cannot competently question the vendor's work.
- Vendors never understand a system and its infrastructure as well as the insiders.
- There is a possibility of us-versus-them conflicts.
- The potential re-use of internal testware is low, such as existing feature test cases which could be adapted for load testing.
- There are potential issues of security and confidentiality.
- System probably will be migrated and re-tuned in the live environment after the vendor's testing, perhaps materially changing its performance characteristics.
- Friction at the turnover point from the external testers with their performance claims, to the internal operations people who must meet the SLAs.
- Track record and history of problems. Many test outsourcing projects end in dissatisfaction. Why is yours different?

## *III. Determining The Performance Test Approach*

**OUTSOURCING WORK SHEETS**

| **Advantages of Outsourcing** | Importance (*) | Presence (**) | Combined Score |
|---|---|---|---|
| 1. Lower initial investment | | | |
| 2. Shorter project elapsed time | | | |
| 3. Lower risk | | | |
| 4. Outsourcing vendor's in-depth expertise | | | |
| 4. Another voice of advocacy (the outsourcing vendor's) | | | |
| 5. Better scalability in the vendor's test lab | | | |
| 6. More objectivity and credibility of the outsourced test results | | | |
| 7. Other advantage (name: ) | | | |
| 8. Other advantage (name: ) | | | |
| Total scores | | | |

(*) Relative importance of this factor in this culture and environment, on a scale from 0 (irrelevant) to 3 (critical).

(**) Degree to which this factor is present in this project, on a scale from 0 (not present) to 3 (pervasive).

## *III. Determining The Performance Test Approach*

| **Disadvantages of Outsourcing** | Importance (*) | Presence (**) | Combined Score |
|---|---|---|---|
| 1. Higher eventual costs | | | |
| 2. Failure to build and establish the expertise in-house | | | |
| 3. Client does not have the competence to question the vendor's work | | | |
| 4. Lack of vendor understanding of the system and its infrastructure | | | |
| 5. Potential for us-versus-them conflicts | | | |
| 6. Low re-use of existing testware | | | |
| 7. Security and confidentiality issues | | | |
| 8. System migration and re-tuning after the vendor's testing | | | |
| 9. Friction at the turnover from the external testers to the internal operations people | | | |
| 10. Track record and history of problems | | | |
| 11. Other disadvantage (name:) | | | |
| 12. Other disadvantage (name:) | | | |
| Total scores | | | |

(*) Relative importance of this factor in this culture and environment, on a scale from 0 (irrelevant) to 3 (critical).

(**) Degree to which this factor is present in this project, on a scale from 0 (not present) to 3 (pervasive).

## *Exercise 2.6: Selecting the Methods of Testing*

### Instructions

There are many ways to measure performance. Selecting the right ones for a particular situation is essential for developing an effective test strategy.

First, review the descriptions of test methods in Appendices A and E. The types and categories overlap to some extent.

Second, rank (a) the usefulness for this project and (b) the likely ease of use of each of the following methods in the work sheet below.

Third, review the entries you made in your work sheet and see which techniques rank highest on both scales, and which rank lowest.

Fourth, question if the work sheet below the best mechanism to facilitate selecting the test methods. How do you suggest we can improve it? Consider these points:

- All test design decisions are influenced by value and cost. To decide what you want to test, you need to know how to think about the value of each one of the test ideas and the cost to do it. A complicating factor is the value and cost of the any test depends on what OTHER testing has already been done, and on the risk profile of the product.

- One way to handle this is to organize your lists by the kind of risk information that is revealed by each family of tests. Describe the value and cost of each kind of testing within that family. Then decide what test strategy to use by evaluating which risks exist in the project. For high risks or when cost is no object, choose more types of testing within each family (because a variety of methods is much stronger testing than using only one), otherwise, less are chosen (or none at all).

- Some kinds of testing reveal information about a very broad array of risks (scenario testing or user testing, for instance) but what they reveal is not necessarily deep information. Other types of testing, such as record/playback style load testing, might provide information that is deep and quantitative, but applies to only one kind of risk.

## *III. Determining The Performance Test Approach*

**Test Methods Work Sheet**

| Test or Measurement Method | Usefulness for this Project | Likely Ease of Use in this Situation |
|---|---|---|
| 1.0 Testing which is driven by what we want to measure. | | |
| 1.1 Response time testing | _____ | _____ |
| 1.2 Throughput testing | _____ | _____ |
| 1.3 Availability testing | _____ | _____ |
| 1.4 Measurement of resource utilization | _____ | _____ |
| 1.5 Capacity testing | _____ | _____ |
| 1.6 Error rate measurement | _____ | _____ |
| 2.0 Testing which is based on the source or type of the load. | | |
| 2.1 Usage-based testing | _____ | _____ |
| 2.2 User scenario testing | _____ | _____ |
| 2.3 Standard benchmark testing | _____ | _____ |
| 2.4 Load variation testing | _____ | _____ |
| 2.5 Ramp-up testing | _____ | _____ |
| 2.6 Component-specific testing | _____ | _____ |
| 2.7 Calibration and iterative feedback testing | _____ | _____ |

## *III. Determining The Performance Test Approach*

3.0 Testing which seeks to stress the system or find its limits.

| | | |
|---|---|---|
| 3.1 Scalability testing | _____ | _____ |
| 3.2 Bottleneck identification and problem isolation testing | _____ | _____ |
| 3.3 Duration or endurance testing | _____ | _____ |
| 3.4 Hot spot testing | _____ | _____ |
| 3.5 Spike and bounce testing | _____ | _____ |
| 3.6 Breakpoint testing | _____ | _____ |
| 3.7 Degraded mode of operation testing | _____ | _____ |
| 3.8 Physical environment testing | _____ | _____ |

4.0 Concurrency testing

| | | |
|---|---|---|
| 4.1 Rendezvous testing | _____ | _____ |
| 4.2 Synchronization testing | _____ | _____ |
| 4.3 Interaction / interference testing | _____ | _____ |
| 4.4 Deadlock testing | _____ | _____ |

5.0 Risk-based testing (*)

| | | |
|---|---|---|
| 5.1 Risk assessment (*) | _____ | _____ |
| 5.2 Hazard or threat identification | _____ | _____ |
| 5.3 Disaster recovery testing | _____ | _____ |

5.4 Bad day testing _____ _____

5.5 Soap opera testing _____ _____

6.0 Testing which focuses on the impact of changes.

6.1 System impact assessment _____ _____

6.2 Infrastructure assessment _____ _____

6.3 Baseline testing _____ _____

6.4 Volume or parallel testing _____ _____

6.5 Live patch testing _____ _____

6.6 Configuration testing _____ _____

6.7 Extreme configuration testing _____ _____

(*) Risk-based is included here for completeness, but it is sufficiently important and complicated that it justifies more attention. The next exercise (2.7) explores risk-based testing further.

## *Exercise 2.7: Determining the Test Focus and Coverage*

This exercise expands on the ideas discussed in the previous exercise, and places risk at the center of the test selection process.

**Instructions**

Perform a risk assessment in order to identify where to focus the deep, intense testing and where else to skim and test lightly, in order to conserve resources for the areas which require the deep testing. You can assess and pinpoint the risks by answering the following series of questions. You may do this exercise individually or in a team brainstorming session.

## *III. Determining The Performance Test Approach*

### QUESTIONS TO ADDRESS (A: PRIMARY FACTORS)

(2.7.1) What circumstances are likely to cause heavy demand on the system from external users (i.e., remote visitors to the web site, who are not book club employees)?

(2.7.2) Under what circumstances is heavy internal demand likely (i.e., by the book club employees)?

(2.7.3) What uses of the system are likely to consume a high level of system resources per event, regardless of how frequently the event occurs? The resource consumption should be significant for each event, not high in aggregate simply because the event happens frequently and thus the total number of events is high.

(2.7.4) What system uses are timing-critical or timing-sensitive?

(2.7.5) What uses are most popular, i.e., they frequently happen?

(2.7.6) What uses are most conspicuous, i.e., have high visibility?

(2.7.7) Based on your understanding of the system architecture and support infrastructure, where are the likely bottlenecks?

(2.7.8) What specifically is new or changed in the coming version of the system or its support infrastructure? Areas which are new or modified are more likely to have performance issues than areas which have already been running satisfactorily and have not been touched. However, if most or all of the system is new, everything is at risk and answering this question will not help.

(2.7.9) What has been your prior experience with other similar situations? Which features or systems aspects typically have encountered performance problems? If you have no experience with other similar systems, please skip this question.

(2.7.10) Are there any notably complex functions in the system, for example, in the area of exception handling?

You are now about halfway through this exercise – time for a brief break to let your brain cells cool off.

### QUESTIONS TO ADDRESS (B: SECONDARY FACTORS)

## *III. Determining The Performance Test Approach*

(2.7.11) Are there any areas in which new and immature technologies have been used, or unknown and untried methodologies?

(2.7.12) Are there any other background applications which share the same infrastructure and are expected to interfere or compete significantly for system resources (e.g., shared servers)?

(2.7.13) What is the architects' and developers' level of confidence in the system's adequacy? Do we know where in the system these people feel comfortable that performance will not be an issue, and in which areas are they nervous? I am assuming that the testers have access to the architects and designers – if not, this question and the next one may be unanswerable and thus irrelevant.

(2.7.14) What are the architects' and developers' reputations for delivering systems which fail to meet the performance goals, and their credibility in spotting potential problems? Since these people usually understand the system internals better than anyone else, their suggestions could be invaluable – but only if they know what they are talking about.

(2.7.15) What can we learn from the behavior of the existing systems that are being replaced, such as their work loads and performance characteristics? How can we apply this information in testing the new system?

(2.7.16) What areas of the system operation, if they have inadequate performance, most impact the bottom line (revenues and profits)?

(2.7.17) What combinations of the factors, which you identified by answering the previous questions, deserve a high test priority? What activities are (a) likely to happen concurrently, and (b) cause heavy load and stress on the systems?

(2.7.18) What areas of the system can be minimally tested for performance without imprudently increasing risk, in order to conserve the test resources for the areas which need heavy testing?

(2.7.19) In summary, considering the total picture, what areas should the performance test focus on? Consolidate your answers to the prior questions in this exercise to form an answer for this question, by completing a table like this:

## *III. Determining The Performance Test Approach*

| **Area to be Tested** | **Likelihood or Probability of Performance Problems in this Area** | **Likely Cost or Consequences of the Performance Problems** | **Exposure (Combined Importance of Likelihood and Consequences)** | **Relative Ease of Testing in this Area** | **Test Priority for this Area** |
|---|---|---|---|---|---|
| Database maintenance (updates, re-indexing, and back-ups) | High (5), because this is a highly data-dependent system with a data-centric system architecture. | High (5), because the database performance affects all parts of the system operation and is highly visible. | High (5), as this is based on the combined effect of the entries in the two columns to the left. | Moderate to high (4), as automated test cases already are available, and a tool is being acquired to run them. | High (5), as this is based on the combined effect of the entries in the two columns to the left. |
| Denial of service (DOS) attack | Low (1), because an attack is assumed to be unlikely. | High (5), because without adequate DOS controls an attack will shut the system down. | Moderate (3) | Moderate to low (2), because a large test load must be generated and delivered. | Moderate (3) |
| Incoming telephone calls to the call center, after a promotion | Moderate to high (4), but expected to decline over time as more people switch to directly ordering via the Internet. | Moderate to high (4), because people are sensitive to phone delays. Members will be irritated and business lost. However, non-telephone work is not affected. | Moderate to high (4) | Low (1), because a small army of testers are needed to manually place phone calls, or specialized, expensive call generators. | Moderate to high (4) |

## *Exercise 2.8: Calculating the Test Work Load*

### Instructions

Answer these questions, based on (a) the following definitions of the terms used here, and (b) the following set of measurements and assumptions. If necessary, you also can refer back to the data presented in Section 2.A, Background Description of the Situation, of this case study, but this exercise is intended to be self-contained. All the information you need is attached in the next couple of pages.

### Questions

#### A. TEST WORK LOAD VOLUMES

(2.8.1) How many web site visits or sessions are expected per month? (Hint: look at assumption 1 below.)

(2.8.2) How many web site visits or sessions are expected in a typical hour? In this calculation, ignore the little-visited hours from midnight to 10.00 am. (Hint: use your answer from question 1 above, and also look at assumptions 2 and 3.)

(2.8.3) How many web site visits or sessions are expected in the peak-demand hour of the day, i.e., in the busiest hour in a typical day? (Hint: use your answers from the prior two questions, though this risks a ripple effect as errors propagate. Also look at the assumptions.)

(2.8.4) (4.8.4) Should we test for the highest peak that can ever occur, or the most likely daily peak (the mode) or the average peak (the mean).

(2.8.5) Before you try calculating your answers to the next few questions, first sketch a graph to help understand the nature of the traffic flow:

l) In this graph, set the horizontal or x axis to represent the time on the clock, with the left edge of the graph being time zero and the right edge being one hour later.

m) Set the vertical or y axis to indicate the amount of activity (e.g., the number of concurrent sessions).

n) Select any time at random within this hour of activity, as the start time for a particular session.

o) Select the length of this session at random – according to the assumptions, the average length is 15 minutes and the range is from 3 to 45 minutes.

p) Draw a horizontal bar on this graph to represent the session. The bar will stretch from the start time for the extent of the session until its end time.

q) Repeat this process until you have a dozen or so bars, and stack each new bar on top of its predecessors in the graph (with a little separation gap between each pair of bars).

r) Remember to include some bars for sessions which were already active before the hour (i.e., their start times are to the left of the zero line), and for sessions which do not end within the hour (these trail off to the right).

s) Select a random point in time and draw a vertical line at that point on the graph.

t) See how many horizontal bars intersect this line.

u) The number of concurrent sessions at this point in time is represented by the height of the pile of bars at that point, in other words, by the number of bars intersecting the vertical line.

v) Experiment with your graph. Draw another couple of vertical lines at other points, and count how many concurrent sessions are occurring at those times.

(2.8.6) What is the expected average number of concurrent visitors at any point in time?

(2.8.7) What is the expected peak number of concurrent visitors within a typical hour?

(2.8.8) What is the expected peak number of concurrent visitors in a typical month?

(2.8.9) Exactly what do we mean by the word “concurrently”? Do all users have to active at any given instant, or can they be waiting for the system and vice versa? What about someone who aborted but did not bother to log out? Until the system times out, should we count them? Do we care either way?

(2.8.10) What is the minimum number of concurrent web site visits or sessions?

(2.8.11) How many hits and page views will happen to the home page in a typical hour?

(2.8.12) How many page views will happen to the home page in the peak hour of a typical month?

(2.8.13) Are there any inconsistencies in your calculations? Can you use dimensional analysis to find questionable answers? (This technique is explained later, in Appendix A.)

So what? How could and should we use the results of these computations in our test work load planning?

You are now about halfway through this exercise – time for a brief break to let your brain cells cool off.

## B. TEST EXECUTION LOGISTICS

What should be the sample size? I.e., about how many of each significant event or transaction should we include in each test run?

How should we validate and cleanse the collected data?

Should we compute and use the mean, median or mode of the data observations?

Approximately what should be the elapsed time for each test run?

During this period, what can we monitor in order to ensure that the measurement process is proceeding in a satisfactory manner?

How do we make decision, if necessary, on whether to continue or abort a test run?

Can we deliberately accelerate the test throughput, faster than real world rate of events occurring, in order to shorten the test duration? If so, how can we adjust for this acceleration of the testing in the data analysis?

Can we make the test case mix more negative (i.e., more destructive) in order to enrich the opportunities for system failure, faster than in the real world rate, in order to shorten the test duration? If so, how can we adjust for this in the data analysis?

## *III. Determining The Performance Test Approach*

Approximately how many cycles of [ test – tune or debug – then re-test ] should we allow for in the project schedule?

What are the main factors which will influence the test duration and number of cycles?

What other significant assumptions besides the ones listed below, if any, have you made in this exercise?

## *Description of the Situation, Section 2.C: Volumetric Assumptions*

### C.1 MEASUREMENTS AND ASSUMPTIONS

Use the following data to calculate your answers to the work load questions. Please document any additional assumptions you make.

#### C.1.1 USER DEMAND

1. The 100,000 active users (book club members) will visit the web site, on average twice per month. Another 15,000 casual visitors (non-members) will access the site per month.

2. The book store is open every day of the month (30 days on average), and the volume of traffic is approximately the same from day to day.

3. This site is visited mostly by U.S. residents, and primarily for personal use, not business use. The hours of most common use are 10 am EST through 12 pm PST (or 14 hours a day). 90% of visits occur during this period, as not many people use the system at 3 am.

#### C.1.2 SESSION STATISTICS

4. A site visit, or session, lasts an average of 15 minutes.

5. While session lengths can be indefinitely long, 95% of sessions lengths fall in the range from 3 minutes to 45 minutes.

6. For the purpose of this exercise, we can ignore the 5% of sessions which last less than 3 minutes or more than 45 minutes.

7. Since calculating the number of overlapping sessions at any given point in time -- what many people call concurrent users – is complex, we can assume that this approximation is sufficiently accurate: the number of concurrent users is equal to the number of hourly sessions multiplied by the average session length (in hours). For example, if there are 600 hourly users and the average session length is 10 minutes, then the number of concurrent sessions is approximately 100 (600 multiplied by 1/6). At any particular instant within that hour, the actual number of concurrent users can vary from zero to 600, but the average during the hour is assumed to be 100.

### C.1.3 HITS AND VIEWS

8. Each visit or session accesses and downloads the home page once on average.

9. Viewing the home page requires 5 hits.

10. Viewing the other pages requires 5 hits per page on average, though the range varies from 1 to 10 hits per page.

### C.1.4 PEAK LOADS

11. In a typical day, the peak hour traffic as measured by the number of hits or page views is 20% of the total day's traffic, including all 24 hours.

12. The number of sessions in the peak hour is expected to be 3 times the number in the average hour. (This assumption considers only the 14 hours from 10.00 am to midnight, and ignores the other 10 hours in the day.)

13. The peak number of concurrent users in a peak hour is expected to be 3 times the peak number in the average hour of the day. (This assumption considers only the 14 hours from 10.00 am to midnight, and ignores the other 10 hours in the day.)

14. The peak number of concurrent users in a typical week is expected to be twice the number in the daily peak hour, or 6 times higher than the number in the average hour.

15. The peak number of concurrent users in a typical month is expected to be 3 times the daily peak hour, or 9 times the average hour.

## *Exercise 2.9: Balancing Exploratory and Structured Testing*

### Instructions

The purpose of this exercise is to determine how much the testing should be structured, with pre-planned test scripts, versus exploratory, where we learn as we go. Commonly, several unknowns complicate performance testing projects and limit the amount of pre-planning – the beginning of the testing is when we know the least about the situation. (See later for a description of the terms exploratory and structured testing.)

Review each of the following items and assign a score for your project to that item, on a scale of 0 (the item is thoroughly understood and largely known) to 5 (the item is unknown or highly uncertain). Then sum the individual scores to obtain a total for the whole list. Typically, the more common uncertainties on performance and robustness testing projects are:

1. How clear, complete, realistic and testable are the performance requirements? _____

2. What aspects of the system's behavior are we interested in examining? In other words, what do we want to measure or monitor? _____

3. How do we analyze and derive conclusions from these measurements? _____

4. What load(s) or mixes of demands can we place on the system while we are measuring its performance and robustness characteristics? _____

5. How can the test equipment be set up, connected and configured? _____

6. What and where are the vulnerabilities in the system we are testing? _____

7. What issues, complications and hassles are we likely to encounter during this performance testing project? _____

8. How comfortable is our team about our ability to handle these issues? _____

9. Are the completion criteria for performance testing well defined, clear and agreed

## *III. Determining The Performance Test Approach*

to? _____

10. How many iterations of debugging, tuning and/or modification will the system have to go through as part of this project? _____

This second group of questions are not as popular as the group above, but they are still widely asked:

11. What is the test process and the set of steps we will be following on this project? _____

12. Who are the users, and how will they use the system? _____

13. How do we use the testing tools we have available? _____

14. What test equipment is needed, and where do we get it from? _____

15. What are we seeking to accomplish with this test project? _____

16. When will the system be ready for performance testing? _____

17. How long will we have for performance testing before the system is released? _____

18. How clear, feasible and agreed to are the performance goals and specifications for the system? _____

19. How unskilled or inexperienced is the test team, and to what extent are they unsure how to proceed? _____

20. What specialized test expertise is needed to bolster the test team, and where do we get it from? _____

Total Score: _____

This number, the total score, is the approximate percentage of the testing effort on the project which should be exploratory, and the remainder of the effort will be structured. This list of questions can be helpful in another way. It allows us to reflect on the situation and identify the project areas which are settled and well understood, versus the areas of uncertainty.

## *III. Determining The Performance Test Approach*

We want to pay more attention, and earlier in the project, to the questions which received high scores. If only a few questions have high scores, it is fairly easy to focus on and manage that handful. If instead most of the questions have high scores, we can't easily manage them all. We'd take the fundamentally different of exploratory testing. If the total score is high (above 65), it is important to inform the managers and clients about the inherently high risk nature of the performance testing project, and to develop contingency plans in case the project does not proceed as expected.

#### HOW MUCH DO WE KNOW?

Structure requires knowledge. Another way to decide how structured the testing can be is to examine how much we know about the situation. Read the section on: "The Information Gathering Checklist". What proportion of this list could you reliably answer at this stage? That is roughly equal to the proportion of the testing that can be pre-planned and structured.

## *Exercise 2.10: Developing Your Test Automation Framework*

#### ASSESSING READINESS FOR TEST AUTOMATION

The purpose of this first part of the exercise is to assess your chances of automating the performance testing successfully.

Please rate these statements as they apply to your current project or to your organization as a whole; for each statement assign a score of between 0 and 10 points (0 -- this item is a major issue; 10 -- it is not an issue):

1. We have the management, developer and user support (e.g., budget, resources, time) we need to proceed successfully with automation. _____

2. We are ready for automation and have a commitment to proceed. _____

3. We understand why we are automating and what benefits specifically will be realized. Our expectations are realistic. _____

4. Our test team has prior experience with automation in the similar environment and with the same or similar test automation tools. _____

5. The pilot project for automation has been selected and is a good choice.

## *III. Determining The Performance Test Approach*

_____

6. The automated testing tools have been selected and are appropriate for the job at hand. _____

7. Our existing test case repositories are re-usable – the automated feature test cases can be converted fairly easily into performance test cases. _____

Total: _____

You may find that this exercise generates more questions than answers -- jot these questions down for follow-up discussion. If you scored 50 or better out of 70 in this quiz, automation is likely to be a breeze for you. If you scored 25 or less, you will probably be in trouble without further preparation.

### DESIGNING THE AUTOMATION FRAMEWORK

The purpose of this exercise is to take an inventory of tools and processes which are already available in your test environment, and to develop a first draft of the automation framework which is required for test automation to succeed.

(2.10.1) Review the test automation framework diagrams in Appendix H, and – for any categories of tools you are not familiar with -- the accompanying tool descriptions in Appendix A. These diagrams present a generic overview of the types of tools often found in test labs.

(2.10.2) Review each type of tool in the framework and answer these questions:

a) Is this test functionality already in place? I.e., is this type of tool already installed and working in the test environment? (Identify the tools which are installed and available, according to the background description. Then map those tools into the types shown in the framework diagrams, to see what types are installed.)

b) If so, is this test functionality working and effective, based on what we are told in the background description?

c) Is this type of tool critical for this performance test project, or somewhat helpful though not critical, or irrelevant?

d) If some tools are already in place which are not primarily performance test tools (e.g., automated functional test tools), can the performance testers "piggyback" intelligently on these?

e) Do the people on your test team already know in depth how to use this tool?

f) Is troubleshooting and tutorial support available on site? Can other internal groups support your team's use of the tool?

g) How well is this tool likely to be integrated with the other testing & QA tools?

h) What data needs to be passed to and from this tool by other tools?

i) Is it better to buy or build this tool for your test environment? Why?

(2.10.3) Draw a block diagram of the automation framework. This diagram should show the tools needed for automation to be effective; indicate which tools are already in place, and show the data flows from tool to tool. If the testing will occur in a multi-platform environment, the diagram should also indicate the platform on which each tool is expected to reside.

Note: we are not looking for a complete, final and polished automation framework at this point, just an initial sketch.

## *Exercise 2.11: Estimating the Number of Test Cycles*

Performance testing is usually highly iterative, with rapid re-tests as bottlenecks are found and resolved. This means the test facility set-up and re-run must be agile. It also means that an early if crude estimate of the number of test cycles is important – if we assume 3 cycles and the project actually requires 15, our deadline is imperiled.

(2.11.1) Approximately how many test cycles are needed?

- To establish a working test facility, including ensuring that test equipment is installed and connected correctly, and debugging test scripts?

- To explore and build a sense of how the system works and the work load flows?

- To uncover and resolve bottlenecks?

- To build confidence that the tested system is ready to go live?

## *Exercise 2.12: Defining the Roles and Responsibilities*

Review the appendix entitled: “Roles and Responsibilities” and answer these questions:

(2.11.1) Who should be involved in this testing project, both within and external to the test team?

(2.11.2)What skills do you need to be represented in the performance test team?

(2.11.3) What should be the performance testers’ roles and responsibilities?

(2.11.4)Who can clarify for you, if necessary, the business requirements and expectations and the performance goals for the system?

(2.11.5)Who can clarify for you who the users are, how they will be using the system, and what their demographics are?

(2.11.6)Who can address your questions about the system design and architecture and their testability?

(2.11.7)Which people can provide you with test data?

(2.11.8)Who are the persons to contact when you find performance problems such as bottlenecks in the infrastructure?

(2.11.9) In which ways do you think the corporate culture is likely to encourage effective testing practices?

(2.11.10) In which ways is it likely to discourage them?

## *Exercise 2.12 Building Flexibility into the Performance Testing*

xxx

## *III. Determining The Performance Test Approach*

## *Exercise 2.13 Coordinating Performance Testing with other Activities*

xxx

# IV. SPECIFYING THE TESTS

## *Exercise 2.14: Developing the Performance Test Scenarios*

A performance test scenario describes a particular test to run, including the mix of demands to place on the system, the test equipment and tools, what data to collect, and so on. (Some people call this a test case, test script, test condition, etc.)

The main purpose of this exercise is to determine how detailed the test planning should be for this project, and how much documentation makes sense in a performance test scenario. A secondary purpose is to learn the attributes of an effective test scenario.

**Instructions**

(2.12.1) Review the list of desirable characteristics of effective scenarios in Section 2.D.

(2.12.2) Form an opinion about the objectives and feasibility of the test scenario below (in Section 2.D), by answering these questions:

a) Are the objectives of this performance test scenario clear and reasonable?
b) Are the objectives worthwhile? Will the scenario deliver information of value?
c) Is the scenario feasible – can it be done without an unreasonable cost, delay, scarce skills and effort?
d) Does the scenario clearly state what skills are required to run it, or is this obvious from the context?
e) What is the likelihood that the tester will need help applying the scenario? Is there any support available, if needed, e.g., in the form of a subject matter expert or a tutorial, on how to use the scenario?
f) Does the scenario provide estimates of the time and effort needed to run it, or are these also fairly obvious?

(2.12.3) Does the scenario provide sufficient direction, so that the tester has what he or she needs to run the test and efficiently produce reliable results?

a) Is there sufficient information provided about the test load? To answer this, attempt to sketch a diagram showing the test load:
    i. On the horizontal axis, show the time in one-minute segments.
    ii. On the vertical axis, show the counts of each major type of transaction (e.g., a catalog announcement, an order).

## *IV. Specifying the Tests*

b) Based on the scenario, will the tester know what test cases to develop or modify, if any?
    i. Manual test cases?
    ii. Automated test cases?
    iii. Test databases?
    iv. Custom test code to be written (e.g., specialized test drivers)?
c) Will the tester know what behaviors to observe, what data to collect, and how to filter or validate that data? Is it clear what accuracy and precision are needed?
d) Will the tester understand how to interpret the collected data and draw conclusions?
e) Will the tester know how to set up the test? (What equipment is needed, how should it be configured, what tools and test case libraries are needed?)

(2.12.4) Are the risks clear and manageable?

a) How do we know that the scenario is being maintained as business and technical conditions evolve, and is up to date?
b) What could go wrong during the testing? Does the scenario identify the tricky areas and describe what can go wrong?
c) Are there adequate warnings and controls over what could go wrong?
d) Have the vulnerabilities and risks of mis-using the test scenario itself been identified and explained in the scenario, plus how to double-check for possibly erroneous conclusions?
e) Will the tester know how to present his or her findings objectively, defend them if necessary, and avoid politics (or work the politics to his advantage), so that the findings and test results receive a fair hearing?

(2.12.5) Is randomization employed effectively? Which parts of this scenario should be randomized?

(2.12.6) What are the pros and cons of implementing this scenario with many distinct and uniquely defined virtual users versus one or a small number of aggregate virtual users, who collectively impose a similar load on the system?

(2.12.7) Is the duration for this test scenario (75 minutes, most of which is spent in data collection) about right? Why not 75 seconds or 75 hours?

(2.12.8) How do you suggest this scenario can be improved?

## *IV. Specifying the Tests*

## *Description of the Situation, Section 2.D: Performance Test Scenarios*

### D.1 A (CLAIMED) HIGH-OPPORTUNITY TEST SCENARIO

This test scenario mimics what happens when a new e-catalog is announced (the announcement is broadcast to book club members). These members respond by browsing through the new catalog on-line or down-loading the catalog, and then place what the managers hope will be a surge of orders. The purposes of this test are to find whether:

(a) the system can handle a peak load,

(b) if it fails under load, it recovers as expected,

(c) there are bottlenecks – particularly in the communications network – at the load levels used in the test, and

(d) a previously untried technology – wireless – is adequate.

This scenario is intended to load and stress the system, in accordance with the test focus and priorities (i.e., with the vulnerabilities highlighted by the risk assessment). The scenario is expected to be realistic, in the sense that it complies with the operational profiles established in the Description of the Situation, Section 2.E.

### D.2 A DETAILED VERSION OF THE TEST SCENARIO

#### D.2.1. Description

Name of this test scenario: *ANNOUNCEMENT OF NEW E-CATALOG*

ID# and version #: The ID# is to be assigned; this is version 0.9 (final internal draft before publication for review by people outside the performance test team).

Area(s) where this scenario is used: Full-system testing, with a realistic peak load.

Cross-references (e.g., to the different customer groups to which this scenario applies, system versions, platforms, etc.): To be added.

Person responsible for this test scenario: You.

Date created or last modified: Today.

**D.2.2. Purpose and Intended Use**

Summary: This scenario determines whether the system can handle the demand that occurs when a new e-catalog is released and triggers a surge of new orders.

Typical work flow (test case):

n this draft scenario, the specific data values for the test work loads have not yet been calculated, and each is designated by “xxx” followed by a unique number (e.g., xxx3). Each “xxx” value later will be replaced by the correct number, as derived from the operational profile. Assigning “xxx” to data values is deliberate, as finding or calculating the actual numbers can take some effort. Instead of mindlessly calculating all possible metrics and then not using many of them, instead we first identify the numbers we need and designate them by “xxx” until we replace them with the actual values.

The event that initiates the test scenario is an announcement of a new e-catalog. The flow of events is as follows:

- Approximately xxx1 people receive the e-mail notification of the new catalog within the first yyy1 minutes.

- In response, xxx2 people download the e-catalog table of contents page within yyy2 minutes of receiving the notification. The periods yyy1 and yyyy2 overlap with the peak number of the table of contents requests occurring yyy3 minutes after the first of the e-mail notifications is received.

- This results in xxx3 additional page downloads from the catalog, xxx4 book searches and xxx5 new orders placed in the next yyy4 minutes.

- Unrelated to the e-catalog announcement, the number of other home page views during the same time are xxx6, and the number of downloaded pages containing sizeable video, audio or graphics files are xxx7. All orders received after this time (i.e., after the first [ yyy1 + yyy2 + yyy4 ] minutes), are ignored in this scenario as the load has diminished from the peak.

- During the same time, the promotion generates xxx8 phone calls to the book club, which in turn generate xxx9 database accesses, queries and updates for

the phone call support and xxx10 additional orders entered by the book club staff. In the same time, xxx11 copies of the paper version of the catalog are printed. (In the test lab, we will simulate this but not actually print catalogs, in order to conserve paper.)

- During the same time, there will be a steady stream of xxx12 timing-sensitive VIP queries and management ad hoc reporting requests, at the rate of xxx8 per hour. Sample query: Compute the net increase of sales in the state of North Carolina from last year to this year, for books selling for more than $50 each. Sample report request: Compute and print a series of 50 reports, one per state within the U.S., of the state sales taxes incurred so far this month.

- Background noise is a part of the test scenario. This noise will be provided by running the billing system and the company e-mail system at normal load levels in the test environment, concurrently with the test scenario.

- Wireless is an unknown and untried technology for the book club and the system developers, so it is included in the test scenario. During the same time, xxx13 messages will be transmitted by wireless to the Warehouse Group, instructing them to pick, pack and ship book orders.

This test scenario is based on the assumption that the first hour after a promotion announcement will be the peak demand hour.

The test scenario is likely to cause a bottleneck in the communications network, if there is one that occurs at these load levels, because it attempts to overload the local area network (LAN) and the external telecom links. The LAN carries both the print requests and database queries and updates. The telecom links carry the voice call traffic and the web site traffic.

[Alternate]

A total of z1 existing book club members receive the e-mail notification of the new catalog's availability, within an elapsed time period of z2 minutes. Of these members, z3 respond within an average of z4 minutes each by downloading the joint cover page / table of contents page of the catalog. The typical respondent requests and reviews z5 more pages of the catalog in the next z6 minutes, and then places a new book order for z7 different book titles z8 minutes later. Orders arriving after the first z9 minutes are ignored in this scenario, since by then the load has dropped significantly below its peak and the system behavior is no longer of interest.

Settings: Test Volumes. Initially, the parameter z1 will be set to 25,000 members (25% of all members), z3 to 10% of z1 (2,500 orders), z5 to3 web pages, and z7 to 1 book title. In later test runs, z1 will be increased to 100,000 and z3 to 15,000.

Settings: Transaction Timings. The values of z2, z4, z6 and z8 are each expected to vary around an average value during a test run. These averages still have to be determined, based on an experiment to observe typical web visitor behavior and an analysis of the operational profile data. Initially, for the purpose of testing the automated test scenario itself, z2 will be set to 15 minutes and the other timing averages (z4, z6 and z8) will be set to 5 minutes each. Note that the responses to the broadcast of the e-mail notifications overlap the broadcast itself, in other words, members do not wait until all notifications have been distributed before the first one responds. The value of z9 initially be set to the sum of the values for z4, z6 and z8.

### D.2.3. Justification

Why this scenario is worth testing: The risk assessment identified this scenario as a high priority – it is relatively likely to occur, and with high-cost consequences when it does occur.

Why this scenario is worth documenting: This level of detail is necessary in order to provide sufficient direction to the test automation specialist(s) who will build and maintain it, and to the testers who will run it.

#### *D.2.3.1Possible Outcomes*

Expected or desirable outcome (pass criteria): The system can handle a volume of x1 catalog downloads and x2 entered orders in an elapsed time period of 30 minutes, while also meeting the performance requirements described below. (These targets, x1 and x2, need to be calculated.)
Anomalies (fail criteria): The system cannot reach and sustain this service level for 30 minutes.

### D.2.4. Target Audience

Who will use this scenario: Performance testers, test automation specialists.

Skills and experience needed to use this scenario: Competence in test set-up, execution and data collection, and results evaluation. Familiarity with the business application.

Programming skills are not needed to use this test scenario, but are needed to create (automate) it.

Who needs or will use the test results: To be added.

Management decision-making needs: Senior management; business unit mangers for the customer service, catalog publishing and warehouse distribution departments; IS managers.

Technical decision-making needs: System developers; support and maintenance personnel; system administrators; database administrators; network engineers; capacity planners.

**D.2.5. Performance Requirements addressed by this Scenario**

Types: Response time; throughput; resource utilization when the system is subjected to peak demand.

Performance targets or goals: Response time of y1 seconds or better; 90% of the time or more; throughput of x1 catalog downloads and x2 orders entered in 15 minutes; with resource utilization of y2% or less of the processor capacity and y3% or less of the memory capacity. Initially, y1 will be set to 3 seconds and both y2 and y3 to 50%.

**D.2.6. Hypotheses to be Proven or Disproven**

Statement of the first hypothesis: The system will pass this test, by meeting or exceeding the performance targets described above, in everyone of a series of test runs, and with adequate documented evidence that it has passed.

Example(s) to prove the hypothesis: Successful completion of a mix of test runs using this scenario. The exact mix has still to be determined.

Counter-example(s) (look for a counter-example to disprove a hypothesis rather than repeated exercises to prove it): One unsuccessful test run. For the test results to be plausible, this test run must be done under realistic conditions. The results also should be unequivocal, not subject to doubt, even if this means devising new variations of the scenario “on the fly” in order to confirm the evidence.

Statement of the second hypothesis: Under the loads planned for this test, the system will still have sufficient spare capacity to handle additional demands.

Proof of this hypothesis: Within the 30 minute duration of each of the test runs, none of the system processors will exceed a processor utilization level or a memory utilization level of 80%, except momentarily (for periods of 30 seconds or less). These thresholds (80% or higher utilization for 30 seconds or less) were determined to be reasonable and prudent goals by the system administrators who will be running the system in live operation.

### D.2.7. Description of the Test Scenario

Overall approach: This scenario mirrors the demands associated with a monthly promotion and announcement of the new e-catalog and the ensuing wave of sales.

Major assumptions: The test work load is a realistic representation of actual demands; the test environment is realistic; the automated test tools are trustworthy and behave as the testers expect (for example, measuring what the testers think they are measuring); the test implementation specialists have built the scenario correctly according to this blueprint (this document); the test focuses on monitoring the key events and does not overlook any significant symptoms; the data collection, interpretation and analysis methods will be adequate to prove or disprove the test hypotheses; the testers have the skills needed to use this scenario correctly; and there is adequate time available to set up and check out the test carefully, and to run and re-run variations of the scenario in order to explore adequately the system's behavior.

Scope: To be determined.

Not Included in the Scope: To be added.

Priority and criticality of this scenario: High.

### D.2.8. Test Infrastructure

Scalability approach – infrastructure realism: The test environment will mirror the live environment, and use a copy of the full live database.

Adjustments for differences between test lab vs. live environment: No adjustments are needed.

#### *D.2.8.1 Test equipment*

Inventory: hardware, networks, databases, support software: Same as used in live

operation, including the wireless.

Equipment preparation: To be added (TBD).

Equipment set-up: TBD.

Configuration: TBD.

Support software and tool installation: TBD.

Data loading: TBD.

Checking readiness: TBD.

People resources: TBD.

Team formation and orientation: TBD.

Timing of resource demands (i.e., forecast of when they are needed during the project): TBD.

Test tools and libraries: TBD.

Types needed: Load generation and measurement – LoadRunner; monitoring tools – to be determined (tbd); profilers – tbd.

Training and tool support needs: LoadRunner.

Test case libraries:
Existing: WinRunner feature test case libraries will be reviewed for the feasibility of their adaptation and re-use by LoadRunner.
To be developed or adapted: An estimated 50% of the test scenarios listed below cannot be adapted but must be built.

Test database(s): A full copy of the live operational database will be used as the test database. Test transactions, whether manual or part of automated test scenarios, must coordinate with or match this database.

Automation framework and harness: None is required other than what is provided by LoadRunner.

Test data sources (e.g., market research): See the case study for operational profile (OP) data.

### 6.2.9. Pre-conditions

Dependencies on other events: The announcement of a new e-catalog.

Loading of test data files: The test database must be loaded.

System calibration: None required

Ramp-up to desired initial state (at start of test): Ramp-up will occur promptly to prepare for the testing, but not fast that the system is not ready for full load. Measurements will not be taken during the ramp-up.

Validation of initial state: The system monitoring capabilities will be used to check the system is functioning satisfactorily in a steady state, with a full normal-case work load being handled.

Triggering event (if any): Broadcast announcement of a new e-catalog.

### 6.2.10. Expected Actions

Narrative description of events that occur during the test period: The announcement leads book club members to either browse through or download the e-catalog, resulting in book searches and new orders.

Flow of events (sequence, dependencies): Natural work flow, e.g., searches before selection and ordering; order status queries after ordering.

Background noise during the test period: On-going book orders, searches and queries at normal levels; billing. e-mail.

### 6.2.11. Timings

Duration of the entire test period: 75 minutes, including two back-to-back test runs,

Test preparation:
Set up: 5 minutes.

Validation of set-up: 2 minutes.
Start up (time to reach a steady state): 5 minutes.
Validation of readiness: 3 minutes.

Main workflow (session durations and data collection intervals):
Part 1: 30 minutes.
Part 2: 30 minutes.

Follow-up:

Test shut down, re-setting the test environment, results logging and clean up: 5 minutes.
Initial review and acceptance of collected test data: 10 minutes.

Justification for these timings: Trial runs; prior experience on similar projects.

**6.2.12. Post-conditions**

Validation of final state: TBD.

Re-setting / restoration ready for next test: TBD.

Logging and archiving of collected data (test results) : TBD.

**6.2.13. Test Work Loads**

Load realism: approach to achieve this.

User biographies and demographics: : TBD.

Work flows
Descriptions
Messages and transactions
Events
Threads and processes
Use cases

Test databases
Demographics (operational profiles)

### 6.2.14. Types of Testing to be Utilized

Response time testing
Throughput testing
Availability testing
Measurement of resource utilization
Measurement of spare capacity
Error rate measurement
Usage-based testing
Load variation testing
Ramp-up testing
Scalability testing
Bottleneck identification and isolation testing
Duration or endurance testing
Hot spot testing
Spike and bounce testing
Breakpoint testing
Rendezvous or synchronization testing
Feature interaction / interference testing

### 6.2.15. Types of Testing that are NOT included in this Project

Standard benchmark testing
Component-specific testing
Calibration testing
Deadlock testing
Degraded mode of operation testing
User scenario, bad day and soap opera testing
Disaster recovery testing
Hazard or threat identification
Compatibility and configuration testing
System change impact assessment
Infrastructure change impact assessment
Baseline testing
Volume testing or parallel testing
Live patch and change testing
Extreme configuration testing

### 6.2.16. Automated Test Scripts Used in this Scenario

## *IV. Specifying the Tests*

Web site demands from external users
- Home page download (mandatory)
- Book search (mandatory)
- Book query (optional for this test scenario)
- Book order (mandatory)
- Credit card authorization (mandatory)
- Query status of existing order (optional)
- Add new member (optional)
- Change membership information (optional)
- Link to another site (optional)

Demands generated internally in response to telephone calls and mail
- Book query (optional)
- Book order (mandatory)
- Credit card authorization (optional)
- Status of existing order(optional)
- Add new member(optional)
- Change membership information(optional)
- Delete existing member (optional)
- Complaint (optional)

Warehouse demands
- Shipping and bill of lading (mandatory)
- Inventory query (optional)
- Publisher query (optional)
- Publisher order (optional)
- Inventory update (optional)
- Order fulfillment: picking instructions and confirmations (mandatory)

Maintenance traffic
- Internal e-mail traffic -- among all departments (mandatory)
- System administration
- On-going administration (optional)
- Periodic (e.g., database back-up) (optional)

Senior management transactions
- Ad-hoc query (mandatory)
- Daily on-line status report (mandatory)
- Weekly or monthly status summary (optional)

## *IV. Specifying the Tests*

### 6.2.17. Manual Test Scripts

Manual versions are needed for all the automated test cases listed above.

### 6.2.18. Data Collection Plans

Inventory of what to measure

Descriptions of how to measure

Conditions / criteria for measurement

How to capture the measurements

Where to log the measurements

Data cleansing and validation

Sample sizes

Sampling rates and techniques

### 6.2.19. Data Interpretation and Analysis Plans

Data interpretation strategy and methods

Expected results (tie these back to targets)

Pattern recognition

- Correlations and comparisons
- Routine behavior
- Interesting anomalies

### 6.2.20. Results Evaluation and Reporting

- Proof or disproof of hypotheses

Derivation of findings and conclusions

Development of recommendations

Presentation and justification of recommendations

### 6.2.21. Proof of Concept / Trial Run

Required? Yes

Assigned to: Not yet assigned

Due date: Not yet scheduled

### 6.2.22. Project Management and Status Tracking

Major milestones

Schedule

Resource needs

Progress reporting

Quality review process

Budget

## *Exercise 2.14: Selecting the Test Tools*

**Instructions xxx**

## *Exercise 2.15: Using the Test Tools*

**Instructions**

The purpose of this exercise is to identify and analyze tool issues which tend to be important in a performance and robustness test project.

## *IV. Specifying the Tests*

**Questions to Address**

(2.12.1) What testing tools should be utilized?

(2.12.2) How should the tools be linked to provide an integrated test framework?

(2.12.3) How do we generate and drive the test loads? E.g., do we capture a copy of live activity or generate fictional test data? Do we drive the test manually or using automated tools?

(2.12.4) What tool skills are required for this performance test? What kinds of people should be on the performance test team?

## *Exercise 2.16: Collecting the Performance Data*

**Questions to Address**

(2.13.1) What data do we want to collect or measure, in order to evaluate the performance of this system? (In other words, how do we define performance for this system?) Should response time, throughput, availability, error rates or resource utilization be measured, some mix of these, or other characteristics?

(2.13.2) Where – at what points or locations -- will the measurements be taken? (In other words, at what points internally within the system or externally to the system do we want to gather data or monitor the system's behavior? What performance data needs to be captured at which locations? We do not need a precise answer yet, just a general sense of where to look)

(2.3.3) What loads or mixes of demands should we place on this system, while measuring its performance, such as an average load, a peak load or an overload?

## *Exercise 2.17: Analyzing the Performance Data*

**Questions to Address**

How should the collected data (the test results) be analyzed and interpreted?

How do we determine, based on the measurements, whether the performance objectives of the system are likely to be met in real-world operation of the system?

## *IV. Specifying the Tests*

Watch this space – this exercise is coming soon.

## *Exercise 2.18: Identifying and Reviewing the Outstanding Issues*

### Questions to Address

What are the likely difficulties involved with this performance test project? What are the risks, if any, that the performance test will not deliver trustworthy or usable results?

What additional information, if any, do you need to know before you can develop an adequate performance test strategy?

What assumptions have you made about this situation? Do these assumptions need to be validated, and if so, how can we validate them?

How should we handle the performance-related system requirements, in areas like usability, security and maintainability, which nominally are outside the scope of the performance testing?

## *Description of the Situation, Section 2.E: Supporting Information*

This section addresses these topics:

- E.1 System Usage Demographics
- E.2 Feature List and Operational Profile
- E.3 Transaction Lengths
- E.4 Other Systems on the Shared Infrastructure
- E.5 Growth Projections
- E.6 Changing Mix of Demands
- E.7 Service Level Agreements
- E.8 System Development and Feature Testing Methodology
- E.9 Automated Test Facilities
- E.10 Test Conditions and Constraints
- E.11 Estimated System Usage

### E.1 SYSTEM USAGE DEMOGRAPHICS

The transactions processed by the order processing and related systems are listed below, together with their expected frequencies of utilization. The transaction volumes

are based on actual counts of transactions within the existing system, which have been adjusted for the expected differences between the existing system and the new one. (There is not always a one-to-one relationship between the transactions in the existing system and the new one.) These counts reflect today's use and do not include the project growth in the future, and do not reflect the anticipated shift from primarily internal ordering (by book club employees) to primarily external (by book club members and visitors through the web site).

### E.1.1 TIMING OF THE OCCURRENCES OF PEAKS

The peak hours of demand for the business groups may not all happen at the same time. Information about when the peak demands occur during a week or month (i.e., at which specific hours), for each feature and type of system use, is not yet available.

## E.2 FEATURE LIST AND OPERATIONAL PROFILE

### E.2.1 CUSTOMER SERVICE GROUP

| User Group / Feature or Transaction | Frequency of Utilization (Transactions per hour) | | | Priority (*) |
|---|---|---|---|---|
| | Normal use | Peak use (**) | | |
| | | Week | Month | |
| Web site page views by external users (direct external demands). These transactions are initiated from external users via the Internet: | | | | |
| Main or home page | 500 | 1,800 | 3,600 | 1 |
| Book search | 250 | 1,000 | 2,000 | 2 |
| Book query (availability, price) | 250 | 1,000 | 2,000 | 1 |
| Book order | 50 | 250 | 500 | 2 |
| Credit card authorization | 50 | 250 | 500 | 2 |
| Query status of existing order | 25 | 50 | 75 | 3 |
| Add new member (sign up) | 5 | 15 | 50 | 1 |
| Change membership information | 5 | 15 | 50 | 3 |
| Link to another site | 125 | 500 | 1,000 | 2 |
| E-mails to / from members and visitors: included later | -- | -- | -- | 1 |
| Totals (rounded) | 1,250 | 5,000 | 10,000 | |

The numbers have been rounded for convenience and thus do not add up exactly.

(*) Priority of each type of transaction, where 1 is the highest and 4 is the lowest priority.

(**) These peaks represent the expected load during the worst-case hour in a typical week, and in a typical month, in the two columns respectively. The peaks for all these types of transactions are not expected to occur within the same hour of the week or the month.

| User Group / Feature or Transaction | Frequency of Utilization (Transactions per hour) | | | Priority (*) |
|---|---|---|---|---|
| | Normal use | Peak use (**) | | |
| | | Week | Month | |
| (b) Demands generated internally in response to telephone calls (the average duration of these telephone calls is 12 minutes each). These transactions are initiated from the personal computers in the customer service group: | | | | |
| Book query | 150 | 300 | 600 | 1 |
| Book order | 100 | 200 | 400 | 2 |
| Credit card authorization | 50 | 100 | 350 | 2 |
| Status of existing order | 25 | 50 | 100 | 2 |
| Add new member | 1 | 5 | 10 | 1 |
| Change membership information | 1 | 5 | 10 | 3 |
| Delete existing member | 0.5 | 2 | 5 | 3 |
| Complaint | 0.5 | 2 | 25 | 1 |
| Totals (rounded) | 300 | 650 | 1,500 | |

### E.2.2 CATALOG PUBLISHING GROUP

| User Group / Feature or Transaction | Frequency of Utilization (Transactions per hour) | | | Priority (*) |
|---|---|---|---|---|
| | Normal use | Peak use (**) | | |
| | | Week | Month | |
| Book list | | | | |
| Add book to list of available books | 2 | 50 | 100 | 3 |
| Change book information | 1 | 50 | 100 | 3 |
| Delete book | 1 | 10 | 20 | 3 |
| Add book review | 5 | 25 | 50 | 3 |

| | | | | |
|---|---|---|---|---|
| Change book review | 1 | 5 | 10 | 3 |
| Monthly catalog of available books | | | | |
| Printed catalog (***) | 0 | 0 | 2,500 | 4 |
| Distributed via the Internet (***) | 0 | 0 | 15,000 | 4 |
| Monthly promotional broadcast (***) | 0 | 0 | 100,000 | 3 |

These transactions are all initiated from the personal computers in the catalog publishing group.

(***) These loadings on the system happen only in occasional bursts; normally the transaction traffic per hour is close to zero.

### E.2.3 WAREHOUSE GROUP

| User Group / Feature or Transaction | Frequency of Utilization (Transactions per hour) | | | Priority (*) |
|---|---|---|---|---|
| | Normal use | Peak use (**) | | |
| | | Week | Month | |
| Shipping and bill of lading (Shipping papers and invoice) | 150 | 500 | 1,000 | 3 |
| Inventory management (****) | | | | |
| Inventory query (Internal levels of inventory) | 10 | 50 | 100 | 1 |
| Publisher query (e.g., review of new book titles) | 10 | 50 | 100 | 2 |
| Publisher order | 5 | 25 | 50 | 3 |
| Inventory update (Receipt of new book data from publishers) | 10 | 500 | 500 | 3 |

These transactions are all initiated from the personal computers in the warehouse group.

| | | | | |
|---|---|---|---|---|
| Order fulfillment | | | | |
| Picking instructions and confirmations | 250 | 750 | 1,250 | 2 |

These transactions are all initiated from the hand-held wireless devices in the warehouse group.

(****) These transactions are not processed by the ordering system, but by other systems which share the same resources.

### E.2.4 INFORMATION SYSTEMS GROUP (****)

| User Group / Feature or Transaction | Frequency of Utilization (Transactions per hour) Normal use | Peak use (**) Week | Peak use (**) Month | Priority (*) |
|---|---|---|---|---|
| System development and maintenance | 50 | 250 | 500 | 2 |
| Internal e-mail traffic -- among all depts. | 1,000 | 3,000 | 5,000 | 2 |
| External e-mail traffic | 50 | 100 | 250 | 1 |
| System administration | | | | |
| On-going administration | 50 | 250 | 500 | 1 |
| Periodic (e.g., database back-up) | 0 | 200,000 | 200,000 | 4 |

These transactions, except the e-mail, are all initiated from the personal computers in the IS group. The e-mail traffic is distributed equally across all the clients in the network.

### E.2.5 SENIOR MANAGEMENT GROUP

| | | | | |
|---|---|---|---|---|
| Ad-hoc query | 0 | 250 | 500 | 1 |
| Daily on-line status report | 10 | 100 | 250 | 2 |
| Weekly status summary | 5 | 500 | 500 | 4 |
| Monthly financial and ops. data (****) | 0 | 0 | 500 | 4 |

These transactions are all initiated from the personal computers in the senior management group.

### E.3 TRANSACTION LENGTHS

Each transaction length in this table includes both the input and response lengths.

### E.3.1 Customer Service Group

| Transaction | Length in kilobytes (KB) Median | Average |
|---|---|---|
| Main or home page download | 2 | 2 |
| Book search | 16 | 32 |
| Book query (availability, price) | 2 | 2 |
| Book order | 2 | 4 |
| Credit card authorization | 2 | 2 |
| Query status of existing order | 2 | 4 |
| Add new member (sign up) | 4 | 4 |
| Change membership information | 4 | 4 |
| Link to another site | 2 | 6 |
| Delete existing member | 2 | 2 |
| Complaint | 2 | 3 |

### E.3.2 Catalog Publishing Group

| | | |
|---|---|---|
| Book list | | |
| Add book to list of available books | 2 | 4 |
| Change book information | 2 | 4 |
| Delete book | 2 | 2 |
| Add book review | 2 | 4 |
| Change book review | 2 | 4 |
| Monthly catalog of available books | | |
| Printed catalog | 512 | 1,024 |
| Distributed via the Internet | 64 | 96 |
| Monthly promotional broadcast | 2 | 3 |

### E.3.3 Warehouse Group

| | | |
|---|---|---|
| Shipping and bill of lading | 2 | 3 |
| Inventory management | | |
| Inventory query | 2 | 2 |
| Publisher query | 2 | 3 |
| Publisher order | 4 | 6 |

| | | |
|---|---|---|
| Inventory update | 2 | 3 |
| Order fulfillment | | |
| Picking instructions and confirmations | 2 | 3 |

### E.3.4 Information Systems Group

| | | |
|---|---|---|
| System development and maintenance | 16 | 20 |
| Internal e-mail traffic -- among all depts. | 1 | 2 |
| System administration | | |
| On-going administration | 2 | 3 |
| Periodic (e.g., database back-up) | 1 | 1 |

### E.3.5 Senior Management Group

| | | |
|---|---|---|
| Ad-hoc query | 2 | 3 |
| Daily on-line status report | 2 | 2 |
| Weekly status summary | 4 | 4 |
| Monthly financial and operating data | 8 | 8 |

## E.4 OTHER SYSTEMS ON THE SHARED INFRASTRUCTURE

### E.4.1 FREQUENCY OF UTILIZATION

The ordering system interacts and shares resources with other systems, such as billing, publisher ordering, market data analysis (data mining) and communications (e-mail). The extra work loads include overhead and background transactions like e-mail between the book club employees, and features which are not part of the ordering system but which are hosted on the same application servers. (Other systems which are not listed here have their own separate application servers, and these are not included in the equipment inventory in the Description of the Situation, Section 2.B.)

The work loads for the features of the other systems which run on the shared servers are as follows:

| User Group / Feature or Transaction | Frequency of Utilization (Transactions per hour) | | Priority (*) |
|---|---|---|---|
| | Normal use | Peak use (**) | |
| Week | | | |
| Month | | | |

#### E.4.1.1 Billing Group

| | | | | |
|---|---|---|---|---|
| Generate e-bill | 50 | 250 | 1,500 | 1 |
| Print bill | 50 | 250 | 500 | 1 |
| Generate reminder notice | 10 | 25 | 250 | 1 |
| Process payment | 50 | 250 | 500 | 2 |
| Adjust bill or payment | 10 | 25 | 150 | 1 |

#### E.4.1.2 Publisher Ordering Group

| | | | | |
|---|---|---|---|---|
| Order from publisher | 10 | 25 | 150 | 1 |
| Receive and store | 10 | 25 | 150 | 1 |
| Pay publisher | 10 | 25 | 150 | 2 |
| Adjust order or receipt | 1 | 10 | 50 | 2 |

#### E.4.1.3 Marketing Group

| | | | | |
|---|---|---|---|---|
| Data mining query | 10 | 25 | 250 | 5 |

### E.4.2 Transaction Lengths

Each transaction length in this table includes both the input and response lengths.

| Transaction | Length in kilobytes (KB) | |
|---|---|---|
| | Median | Average |
| Generate e-bill | 2 | 2 |
| Print bill | 1 | 1 |
| Generate reminder notice | 1 | 1 |
| Process payment | 1 | 1 |
| Adjust bill or payment | 2 | 2 |
| Order from publisher | 2 | 2 |
| Receive and store | 2 | 2 |
| Pay publisher | 1 | 1 |
| Adjust order or receipt | 1 | 1 |
| Data mining query | 2 | 8 |
| E-mail | 1 | 2 |

## E.5 GROWTH PROJECTIONS

The system must be scalable or upgradeable to support book club member growth rates of 15% per year, and order volume growth rates of 25% per year, for the next four years.

The growth is expected to be achieved primarily by providing an enhanced web site for book club members and potential members. For this reason, the mix of books sold over the Internet versus through the traditional channels is expected to change, as follows:

| Time Frame | Percentage of Books Sold | |
|---|---|---|
| | By Traditional Channels | By Internet |
| This year | 70% | 30% |
| 1 year ahead | 50% | 50% |
| 2 years ahead | 10% | 90% |

Please note that this table does not include default orders. By default, members receive the monthly book selection without placing an explicit order. A book club member must state explicitly that he or she does not `want the book of the month; otherwise the book is shipped to him.

### E.6 CHANGING MIX OF DEMANDS

The mix of ordering demands is expected to evolve to [10% traditional channels – 90% web], with a corresponding expected shift of work from the telephones to the web. This means that the internally generated work load in response to telephone calls is expected to fall over the next 5 years, not rise, even after allowing for the overall growth in business volumes. So the available or spare CTI (call center) capacity will effectively increase.

### E.7 SERVICE LEVEL AGREEMENTS (SLAS)

A separate service level agreement has been developed for each of the three operational groups, the customer service group, the catalog publishing group, the warehouse distribution group. Each service level agreement has been signed by the manager of the pertinent business group and by the manager of the information systems group. The adherence to the service level agreements will be monitored and reported to senior management monthly, and the information systems group will be evaluated in part on their ability to satisfy the service level agreements.

The following statements summarize the key points in each of these service level

agreements. The agreements state that, for priority 1 features under normal operating conditions (e.g., with an average work load), the internal users' response times should be less than 0.5 seconds, 90% of the time or better, on the client machines. (Priority 1 is the highest priority.) Under peak load, the internal users' response time should be 1 second or less, 90% of the time. This response time goal must be met for each business unit individually, such as the customer service group, the catalog publishing group, and the warehouse distribution group. In other words, it is not sufficient if the average response time across all the business units meets this goal, while the average for any one business does not meet the goal.

For priority 2 features, the internal users' response times should be less than 2 seconds, 90% of the time or better. For features with priority levels 3 or more, no response time guidelines have been established as part of the SLAs.

For the external users who visit the web site, the requirement is that the book club's web site is "noticeably faster" than competitors' web sites. It has been determined that this means that the book club's home page on the web site can be downloaded in no more than 4 seconds on average, under an average work load, and no more than 10 seconds under peak load. These times are measured from when a user clicks to initiate an action until when the user starts to see a response, e.g., when a page starts to be rendered, not when the page is fully visible. These response time targets are the averages for all web site visitors. (A person using a 9.6 Kbps modem will have a slower response than someone with a dedicated T1 line.) The book club also anticipates that these web-based response time targets will need to be revised at least once every 6 months, and tightened, as competitors' web services become progressively faster.

On-going processes will need to be put in place, to monitor and report the level of compliance with the SLAs during the on-going live operation of the system. Developing these mechanisms to monitor and report on SLA compliance is not part of the scope of the performance test project.

The service level agreements address other areas besides response time and throughput, such as acceptable levels of errors in processing, system availability, etc., but these areas are outside the scope of the performance testing.

### E.8 System Development and Feature Testing Methodologies

The new book club order processing system is currently in the process of being developed. Where possible, the developers are re-using existing software components, which have either been purchased from external sources or derived from other internal

systems.

Rapid application development (RAD), which is also called the iterative spiral approach, is being used by the developers. In this approach, the feature testing starts early and overlaps development, and components and sub-assemblies of components are tested as they become available, and then re-tested as the components are debugged or enhanced.

There is a feature testing team, which has been organized to test the features of the new order processing system. This team is not responsible for performance, load or stress testing, but its members have been asked to cooperate and provide assistance to the performance test team, as appropriate and without interfering with the feature test project.

The developers and testers are planning to build component-level test drivers, in order to start unit-testing each major software component as soon as it becomes available.

### E.9 AUTOMATED TEST FACILITIES

This feature testing team is in the process of building a library of automated feature test cases for each of the features listed above. Depending on the criticality, risk and complexity of each feature, the library is expected to contain anywhere from one to ten test cases for each feature.

These test cases are being built and will be executed and maintained by using WinRunner, a tool from Mercury Interactive of Sunnyvale, CA. The feature testing team has purchased five copies of WinRunner, which could be available for performance testing when the feature testers are not using them. Note that the mention of any particular tool in this case study, such as WinRunner, should not be construed to be a recommendation or endorsement of that tool.

### E.10 TEST CONDITIONS AND CONSTRAINTS

At this time, the senior managers have not set a deadline or budget limit for this performance testing project. This does not mean that they will be willing to accept an indefinitely long time frame or an indefinitely large budget for this project, but they are waiting until they hear what you propose.

## *Exercise 2.19: Transitioning to Post-Delivery Live Performance Monitoring*

### Introduction

xxx

## *Exercise 2.20: Team Discussion of the Remaining Test Issues*

### Instructions

Together with your teammates, compare your answers to the previous questions. The intention is not necessarily to reach consensus, though that's fine, but to obtain a deeper appreciation of the issues by seeing others' perspectives. Note that we are not looking for polished and detailed answers at this time, just an in initial sketch of your key thoughts, ready for discussion with the whole class.

# APPENDICES

## APPENDIX A. BASIC DEFINITIONS AND CONCEPTS

Performance testers use a wide diversity of names for the same concepts, and the same word often is used for several different things, indicating the immaturity of the field. We have no universal consistency in how people use terms like performance test and robustness test. I can say that the definitions provided in this book are as much or more in the mainstream as any others.

***Acceptance Criteria*** The criteria to be met before a system or component can be accepted.

***Acceptance Test*** A test to determine if a system meets its acceptance criteria, usually performed by the users, a QA group or an independent third party on behalf of the users, after the system test and prior to accepting the delivered system.

***Accuracy vs. Precision*** The word accurate means correct and on target. The word precise means exact, attentive to detail. Precise is not the same as accurate. Accuracy means correctness but within some tolerance for error, while precision means specificity. If I say the time is now 10.12 am and exactly 36 seconds Eastern Standard Time within New York, I'm being precise to the nearest second but not necessarily accurate. (My watch might be a few minutes off.) If I say that it's now about 10.15 am, I am accurate but not as precise.

***Active / Inactive Status*** The state of an interactive session. During a visit, there can be a string of both active periods (where the system is doing something related to this session) and interspersed inactive periods (where the system is waiting).

***Active Object*** An object that currently is active, i.e., is executing or is waiting ready to be triggered by an event.

***Activity*** A work task or an event.

***Actor*** In UML and object-oriented programming, an object which is active concurrently with other processes and can interact with other actors.

***Ad Hoc Test*** A testing activity where the tester randomly tries the system's functionality. See also monkey testing.

***Adaptive Learning*** The process where software uses feedback to calibrate and adjust weighting factors over time, so that it comes closer to producing the desired outcome.

***ADL*** Assertion (or API) definition language: language used to write assertions and generate test cases for them, or for APIs.

***AES*** Application environment specification: guidelines from the Open Software Foundation for user interfaces, aimed at providing a consistent application environment on different hardware platforms.

***Agent*** A piece of software which runs autonomously, usually to help a person achieve a goal. An agent has its own attributes, and performs tasks on behalf of another entity (another agent or a person).

***Agile Method*** A software development approach which seek to minimize the paperwork, bureaucracy and delays which sometimes occur with traditional development methods.

***Agile Test*** Testing practices on projects using agile methodologies, and emphasizing the ability of testers to respond adroitly to changes.

# *Appendix A: Basic Definitions and Concepts*

***AI*** Artificial intelligence.
***AKA*** Also known as.
***Alert*** A designed-in indicator of a problem with a system in operation. Can take the form of an audible alarm, visual error message, electronic signal, etc.
***Algorithm Test*** An algorithm test helps to ensure that the algorithm selected for a task is the most appropriate one, has been implemented correctly, is stable and comes to closure with an acceptable result, and meets all accuracy, timing, and sizing requirements.
***Algorithm*** A systematic procedure with well-defined rules to solve a problem in a finite number of steps.
***Allocation*** Assignment of a portion of memory by an operating system to a particular application as and when requested by that application.
***Alpha Test*** Usually conducted within a software vendor, the last level of internal test prior to a (beta) release. Roughly analogous to a system test within an IS shop. In a vendor-customer relationship, alpha testing is the last level of testing conducted internally before external customer testing (beta testing) and installation. An alpha test can be performed by the vendor or by a customer in a controlled environment at the vendor's site or a third-party certification site. The software is tested in an environment which is as close as feasible to the live operational environment.
***American Standard Code for Information Interchange (ASCII)*** The primary set of codes used to represent characters in computer systems. ASCII uses a unique seven-bit pattern for each letter of the English alphabet, number between 0 and 9, or special character such as the asterisk.
***Analytical Model*** Uses queuing theory, statistical formulas and algorithms to predict response times and utilization projections from work load characterization data and essential system relationships. Analytic models require input data such as arrival rates, user profiles, and service demands placed by each work load on various system resources. System monitors and accounting logs can provide most or all of the required information. Most analytic models allow for the use of various assumptions in order to keep the solution simple and efficient. The affect of these assumptions on the accuracy of your conclusions must be considered when making recommendations. Analytic modeling is best used when a numeric prediction is needed, when work load forecasts are fairly accurate, when a best-case/worst-case situation holds, or when an accurate answer can save significant money, time, or headaches.
***Anchor*** The location of a hyperlink in a Web page or document.
***Anomaly*** A deviation from expected behavior. See: bug, defect, error, discrepancy, exception, fault.
***ANSI*** American National Standards Institute, which provides many U.S. standards pertaining to software and quality.
***API*** Application program interface: a term for the interface by which an application program gains access to operating system and other services, defined at source-code level.
***Applet*** A small application, often downloaded from a remote server and run in a controlled environment. Typically written in a language such as Java for execution by a Web browser.
***Application*** Software designed to fill specific needs of a user; for example, software for navigation, payroll, or process control. Contrast with support software; system software.
***Application Program Interface (API)*** (1) A set of functions that provide application software developers with access to functionality provided by the operating system or support software. (2) The software components within the operating system which provides the service to the application. (3) A set of software calls and routines that can be referenced by an application program in order to access supporting system or network services. (4) A standard interface between an application and an operating system or other support software.
***Architecture*** The structure and organization of a system. Synonym: design.
***Artificial Work Load*** An artificial work load is used primarily in isolation tests, where the causes of a

particular problem must be diagnosed. The demands in this artificial work load are engineered to force the problem to occur repeatedly.

***ASCII*** American Standard Code for Information Interchange.

***ASIC*** Application-Specific Integrated Circuit: a hardware chip designed and hard-wired to perform a particular function.

***Assembler*** A software development tool which translates assembly language into machine-language instructions which the processor can understand and execute.

***Assertion*** A state which must exist in software or its environment at a particular point during the software's execution. An assertion normally is checked continually during execution.

***Assertion Check*** A statement inserted into software to verify an assertion before continuing with the execution of the software, and to trigger the appropriate action if the assertion is not true. See: instrumentation.

***Asynchronous*** An event which is not, or does not have to be, synchronized with other events or activities.

***Audit*** (1) An examination of a process or outcome to assess its effectiveness, correctness and compliance with standards. (2) An independent review of a system, database, system environment or operational procedure to determine whether it complies with standards or generally accepted practices.

***Auditability*** Can the accuracy and trustworthiness of the system easily be confirmed?

***AUT*** Application under test. Similar to SUT (system under test).

*Automated Test Planner* A tool that provides fill-in-the-blanks templates of testing projects, test plans and test cases.

***Automated testing*** Unattended testing which does not require a person to be present to execute the test cases. The testing is controlled by an automated tool, without the need for human intervention.

***Availability*** This is the percentage of uptime for a system or component over a given duration and under a given load, so testing availability is essentially a process of recording when the system is up or down, under both typical and stress working conditions. Availability is closely related to reliability and robustness. Availability measures can either include or exclude planned downtime, leading to apples and oranges comparisons. Another complication in measuring availability is that many systems can operate in a degraded mode if the need arises, e.g., if part of a network is down the other parts will still function. During this degraded mode, some users may experience limited availability.

***Bachman diagram*** A style of entity-relationship modeling.

***Back propagation*** A method for adaptive learning which is based on the difference between the desired and actual output. The differences are used to adjust the weights assigned to various factors which influence the outcome.

***Back-end*** (1) The back-end of a project includes all activities after the coding is complete (assuming that testing is not done in parallel with coding). (2) The portion of a system architecture which is not externally oriented, e.g., not customer-facing or vendor-facing.

***Backhoe Detector*** A joke name for a telecom cable – the cable attracts the destructive backhoe to where it is buried.

***Back-to-back testing*** See parallel testing, volume testing.

***Backus Naur Form (BNF)*** A formal method for the syntax specification of computer languages.

***Bad Day Test*** This is based on the premise that we all can hit the wrong button or have a bad day. If someone does push the wrong button, we want to ensure that we and they do not suffer any horrendous consequences. Bad day testing is similar to usability testing and operator error testing.

***Bandwidth*** A network's or communication link's bandwidth is the maximum number of bits that can be transmitted over the link in a given time interval, usually one second. Bandwidth determines how much

traffic a network can carry. The bandwidth depends on the physical wire itself and on whatever network device is used. Bandwidth is the amount of information that can be transmitted in a given amount of time through a network connection. Bandwidth is usually measured in bits per second (bps) or cycles per second (Hz).

***Baseline*** (1) A previously tested version of the system, against which a parallel or regression test is known to have passed before the system has been modified. (2) The "before" version of the output results from the previous test, e.g., the response time of the previous version of the system. (3) A version of a system, an operational environment or a test suite which is documented in sufficient detail to re-create it if necessary, provide a basis for further evolution, and can be changed only through a controlled procedure. (4) A controlled earlier version of a system, before changes are introduced. The most common use of the term baseline is the measurement of the performance of an existing system or operation, as the "before" part of a before-and-after comparison. We measure the baseline in a live environment which will disappear (or at least be modified), when the change is implemented. The baseline helps us understand the current behavior if the metrics available are informal, anecdotal ("it runs slowly"), untrustworthy or politicized, and if it is important to later show evidence of the claimed improvements caused by the change. See: release, version.

***Baseline Point*** The point at which a deliverable produced during the software engineering process is placed under formal change control.

***Baseline Test*** This approach measures the performance of an existing system or operation, as the "before" part of a before-and-after comparison which cannot be done after a change has been implemented. We measure the baseline in a live environment which will disappear (or at least be modified), when the change is implemented. We need this measurement to understand the current behavior if the metrics available in the existing situation are informal, anecdotal ("it runs slowly"), untrustworthy or politicized, and if it is important to be able later to show evidence of the improvements (or lack of them) caused by the change. A baseline test may not be the same as the volume and parallel test methods. While we compare the same performance characteristics of the new system with those of the baseline system, e.g., the response time, in a baseline test we do not necessarily re-run the same load with the new system. Often, the new system does not have exactly the same features and transactions as the old one, so we cannot re-use the old load without modification. (Although the load on the new system should be similar to the old load, in order to avoid apples-and-oranges comparisons.)

***Basis Path*** A unique path through a software component in which no repetitions of loops occur (i.e., a loop cannot be taken more than once).

***Batch Processing*** A mode of system operation where a batch of input transactions is assembled prior to execution and processed together, not processed as soon as they arrive. The batch processing, once started, is expected to proceed to completion without interaction. Contrast with event-driven, conversational, interactive, on-line, real time.

***Batch Test*** Batch testing requires the preparation and submission of batches of test transactions, the preparation of test data bases to be accessed and updated by the transactions, and the evaluation of the test results after the complete batch has been processed. On-line processing is normally not part of this batch testing.

***Bebugging*** A method where errors are deliberately introduced in order to provide a "treasure hunt" motivation for testers and debuggers to search for bugs. See error seeding, software fault injection.

***Benchmark*** A standard work load used in testing, instead of one based on an operational profile (OP). Benchmarks are often industry-wide, e.g., the TPC-C and TPC-W benchmarks. Benchmarks are convenient, but the question is how well the benchmark can substitute for an OP – how well it represents reality. Benchmarks are best used to compare two or more systems' behavior under the same load, or different application system versions, or the same system version running on different hardware, and so

on. Any benchmark that has been in existence long enough to become industry-standard has been studied extensively by system vendors: operating systems, compilers, and in some cases hardware have been tuned to run the benchmark with lightning speed.

***Benchmark Test*** (1) A test in which a benchmark mix of demands in run against the system being tested. (2) Testing a system by comparing its behavior to another system (the benchmark).

***Beta Test*** (1) Test where a preliminary version of the system is tried by end users. Beta testing is a vendor's equivalent of a pilot test (usually with a greater number of participating sites than for an in-house pilot), where a pre-release version of a product is distributed on a trial basis. (2) Testing conducted by customers or users on a vendor's product, before the product is widely shipped, installed and placed in live operation by many users. See: field test, pilot test

***Big Bang*** An integration approach where components and subsystems are connected together with little or no integration testing, on order to speed the process and conserve resources for testing of the fully integrated system. Unless the components have previously been integrated and tested to show they work together in prior versions, it is suitable only for small systems.

***Binary*** A variable or a decision which can have one of only two possible values, i.e., 1 (true) and 0 (false).

***BIST*** Built-in system test, usually incorporated into sophisticated electronics such as semiconductor chips.

***Bit*** Binary code which can be either 1 or 0. Virtually all computers process and store data in the form of bits.

***Bitmap*** A format for storing graphics files, usually with the suffixes like .gif or .bmp.

***Black Box*** A view of a system or subsystem that is based on its externally observable behavior rather than its internal structure.

***Black Box Test*** Any test that does not use knowledge of the internal structure of the system, but only checks the system conforms with the expected behavior which is visible externally. Black box testing traditionally has involved testing a system as a whole; however, with the advent of multi-tiered systems, black box testing has come to also refer to testing tiers or components of a total system since those tiers are often entire systems in themselves with only loose ties to the other tiers of the system.

***Boolean*** Variables and operations on variables which are binary.

***Boolean Test*** A test of Boolean computations (i.e., computations that use Boolean logic operators and operands such as and, or, nand, xor) -- these computations are common in aerospace.

***Bootstrap*** To initialize or start up. See start-up code.

***Bottleneck*** Limitation on system performance at a point where there is no spare capacity.

***Bottleneck Identification and Isolation Test*** These activities are really diagnosis and debugging, rather than "pure" measurement and testing, but the performance testers often do them as a service to the developers and system architects. We need these activities because it is very difficult to tune a system without knowing where the bottlenecks are. Generally, we insert invasive probes into the system we are testing and its environment to monitor for bottlenecks. These probes look out for conditions like buffer overflows, a set of ports which are continually busy, bandwidth utilization in a network which is near the maximum available bandwidth capacity, and so on. The performance testers need to have the right test equipment, test tools and expertise to conduct the resource utilization and bottleneck identification activities (more about this later). Sometimes the bottlenecks are obvious and can be quickly found without sophisticated monitoring and diagnosis tools, once the performance test results highlight them and motivate us to go and examine them.

***Bottom-Up Test*** An approach to integration testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the

component at the top of the hierarchy is tested.

***Boundary*** A point at which there is an expected change of behavior in a computer system, such as the upper limit of the allowed range of a variable.

***Boundary Test*** A test which focuses on the boundary values or limit conditions of the software being tested.

***Boundary Value Analysis*** A technique for selecting test data on the boundaries of the input domain or output range. BVA is similar to equivalence partitioning but focuses on "corner cases" or values that are usually just in and out of range, as defined by the specification.

***BPM*** Business process modeling.

***BPR*** Business process re-engineering.

***Branch*** A decision point or a source code instruction which causes the software being run to jump to a new point in the program sequence, rather than execute the next instruction. Synonym: jump, decision point.

***Branch Coverage*** The percentage of software branches covered in testing. For 100% coverage, every branch at each decision point is exercised at least once in testing. Caution: in some definitions, branch coverage is exactly the same as path coverage, while in other definitions the terms branch coverage and path coverage can differ.

***Branch Test*** Testing in which selected branches in the program source code are tested. See path analysis.

***Breakpoint*** (1) The point in a piece of software at which the software engineer starts or stops the software execution during debugging. (2) A location in software at which control of the processor switched to a debugger. (3) The point at which a system fails as its load increases.

***Breakpoint Test*** In this type of stress testing, we increase the load until the system fails. (Or we increase the load as much as is feasible within the constraints of the test environment, if the heaviest load is not sufficient to force the system to fail.) The purpose is to determine the load at which the system breaks, where (at what point within the system) the breakage occurs, and how the breakage is manifested (how the system fails, e.g., a database overflow, or a network link goes down).

***Browser*** A software product which is designed to interface with the World Wide Web and uses HTML files.

Buffer A holding or staging area, generally used as a FIFO queue.

***Buffer or Queue Overflow*** The usual result of an attempt to add an item to a buffer that is full.

***Bug*** An unwanted and unintended property of a system, especially one that causes a malfunction.

***Build*** (1) An integrated collection of software components (modules) which are tested as a group. (2) The process of linking software components to make a build.

*Build & Load Tool* These tools compile and integrate the latest versions of the SUT (system under test) components, integrate related files such as help and configuration files, to form a new executable version of the SUT. The tools also load the SUT on to the test and the live operational platforms.

***Built-in System Test (BIST)*** Any embedded self-test capability of a system.

***Bus*** A set of hardware paths (traditionally electrical lines) which connect hardware components, e.g., links the processor to the peripherals.

***Byte*** A data unit which contains a string of bits but which is smaller than a full computer word. A byte is usually 8 bits.

***C/S*** Client/server.

***Cache*** A small fast memory holding recently-accessed data, designed to speed up further access.

# *Appendix A: Basic Definitions and Concepts*

***Calibration Test*** *This type of testing is interleaved with tuning, and its purpose is to provide feedback on the consequences of each iteration of tuning. The test work load is kept the same, and typically the testers strive for exact repeatability of the test run from iteration to iteration of tuning.*

***Call chain*** The hierarchy or sequence of linkages among software components which work together as part of a system.

***Call Pair*** Two linked software components: the combination of a calling software component and a called software component, where one calls (references or accesses) the other.

***Call Pair Coverage*** Measure of the number of call pairs executed.

***Capability Maturity Model (CMM)*** A method for assessing the effectiveness of software engineering organizations, developed and supported by the Software Engineering Institute at Carnegie-Mellon University (SEI).

***Capacity*** Spare capacity is the room to grow -- the ability of a system to support an additional work load without degrading performance to an unacceptable degree.

***Capacity Test*** This type of testing measures whether the allocated resources are sufficient for the job, how much spare capacity still remains in system for further growth of demand, and at what point in the growth the resources supporting the system will need to be upgraded. This is the point at which the response time or throughput become unacceptable as the demand grows.

***Capture/Replay Tool*** An automated test tool that records test input as it is sent to the software under test. The input cases stored can then be used to reproduce the test at a later time. Most commonly applied to GUI testing.

***Cause-Effect Graphing*** A graphic technique for identifying test cases based on the relationships of the outputs (effects) to the inputs (causes). The cause-effect graphing technique has been automated in commercially available tools, such as SoftTest from Bender Associates, which automatically generate test cases from a description of the functionality or logic. Cause-effect graphing can identify logical inconsistencies and omissions in the functionality, so the method and tools can be used to "clean up" the system requirements definition.

***CCITT*** An ITU committee which sets telecommunications standards and recommends telecommunications systems.

***Certification*** Formal accreditation that a system conforms to defined standards.

***CGI*** Common Gateway Interface, an Internet protocol.

***Change Control*** (1) The organized process for evaluating and taking action on proposed changes to a system or its environment. (2) The process used to control all changes to software, its environment, documentation, test case libraries, and so on. Synonym: change management.

***Change Impact Assessment*** Assesses the impact of a change or a group of changes to an existing system.

***Change Management*** The controlled process of introducing changes into an existing system or component or a system in development.

***Change Request*** The official, documented mechanism for requesting a change.

***Chaos or Perturbation Test*** (butterfly effect) Testing with small changes to the conditions. In chaos theory, a tiny change in the initial conditions or the input data values to an inherently unstable system can cause large differences in outcomes. For software, chaos theory implies that small and apparently insignificant changes can lead to large undesirable changes elsewhere in the system.

***Character*** A symbol such as the letter A which requires one byte of storage.

***Checkpoint*** Interim point in a process where the results to date are checked before proceeding.

***Chen diagram***. A form of an Entity-Relationship data model, named for Peter Chen.

***Child version*** A sub-version derived from a parent version. See change management.

# *Appendix A: Basic Definitions and Concepts*

***Class*** The general description of a family or a group of related objects.
***Class Library*** A library of re-usable classes for use in object-oriented programming.
***Cleanroom*** (1) A dust-free lab or manufacturing facility for electronics and nanology. (2) An inspection-intensive software development approach to produce software with extremely low defect rates. See software cleanroom.
***Client*** (1) A system or process that requests a service from another system or process. (2) The client part of a client/server architecture. Typically, a client is a personal computer or workstation which relies on a server to perform operations. (3) The person or group who is paying the bill for system development and maintenance.
***Client/Server*** A network of connected computing devices which are either servers (these provide services) and clients (these are serviced).
***Client Side Think Time*** The time the browser or other client side application spend processing the data received from a remote server. Must be accounted for when discussing true user experience.
***CM*** Configuration management.
***CMM*** See Capability Maturity Model.
***Code*** Software statements or instructions. See: program, source code.
***Code Complete*** Milestone in software development where functionality is implemented in entirety; bug fixes are all that are left. All functions found in the Functional Specifications have been implemented.
***Code Coverage*** A method to determine which parts of the software have been executed (covered) by a suite of test cases and which parts have not been executed and therefore may require additional attention.
***Code Freeze*** The point after which no changes can be made to a base of source code except to fix severe defects, and then only under strict change management procedures.
***Code Inspection*** A formal review technique where the author (a software engineer) reviews his or her source code with a group of peers.
***Coding*** The process of writing software, or the automated generation of source code.
***Coding Standards*** Written guidelines which define the programming conventions to be used by the software engineers.
***COM*** Common Object Model. A standard from Microsoft to guide the operation and interactions of OLE components.
***Commercial-off-the-shelf software (COTS)*** In the government and military, vendors' application software packages which are pre-built and available for acquisition.
***Common Gateway Interface*** A protocol used by Web-based systems to communicate with their environments.
***Commonality and Variances*** Test technique to develop test cases based on similarities and differences. Group together test situations, based on what they have in common. Two or more situations are equivalent if the successful test result from one situation means that the others would also work successfully and thus do not need to be separately tested. For tests that are equivalent, execute only (a) a representative or random example of all the possible tests, or (b) the worst-case test. Identify the variances that are not covered by these common tests, and ensure these variances are covered in testing also.
***Communicating Sequential Process*** A model of software operations where processes communicate with other through semaphores.
***Communications Test*** The testing of communications systems and networks, or the communications-specific components of more comprehensive systems
***Comparator*** A software tool which compares two versions of data files, such as the versions of a test

results file before and after a modification, and identifies the discrepancies.

***Compatibility*** Does the system operate in the same way across different computer and network configurations, platforms and environments, and with different mixes of other applications? Is it portable? Is it backwards-compatible? Is a new system compatible with the existing technical infrastructure?

***Compatibility and Configuration Test*** This method considers the various configurations in which a system can be used, and how to check for compatibility or consistent behavior across these configurations. (1) Testing for the similarity of system functionality among different versions of a system, or across the different operating environments in which the system runs, or for compatibility of the interfaces to other systems with which the system communicates. (2) Testing whether software is compatible with other elements of a system with which it should operate.

***Compiler*** A software development tool which translates the high-level source code written by software engineers into the machine-language instructions that a particular processor will execute.

*Complexity Analyzer* These tools analyze software complexity, predict defects, and generate test cases based on the structure of the software code, using McCabe's cyclomatic complexity measure or similar techniques.

***Compliance Test*** Testing and comparing a system to a defined standard and determining its degree of compliance with the standard.

Component (software component). (1) A part of a software system which has a distinct purpose and identity, and cannot be decomposed into smaller components, but only into source instructions (lines of code). (2) A minimal software item for which a separate specification is available. Synonym: module.

***Component Metric*** Any measurement related to hardware resource or sub-set of user experience measurements. Some examples include, CPU utilization, RAM usage, database seek time, etc.

***Component Re-use*** The ability to re-use a software component, possibly after adapting it to the new use, or a design pattern, portion of a requirements model, test case, etc.

***Component-Specific Test*** This type of testing examines the performance and robustness of one system component (or sub-assembly). It can be done as soon as the component is ready, before other components are built and well before the fully integrated system is ready for testing. By examining the behavior of one component in isolation, this testing makes it easier to isolate and pinpoint problems. And component bugs which are found earlier can also be eliminated earlier, improving the initial quality of the fully integrated system when it is delivered for testing. Component-specific testing requires component test drivers. These can be expensive to build.

***Compression*** Process of reducing the size of a data file to take less network bandwidth and storage memory.

***Concurrency*** The occurrence of two or more events at the same instant or during the same brief time interval. If events are not simultaneous, to be considered concurrent they should occur closely enough in time to have a non-trivial chance of competing for the same resources, or of interacting and possibly interfering with each other. The occurrence of two or more activities during the same time interval. Concurrency can be achieved by interleaving or simultaneously executing two or more threads.

***Concurrent Users (or Visitors)*** The total number of overlapping users who are actually using the system or who have sessions underway at a specific instant in time. These users are all connected to the system, but may be in different operating states (ready, waiting for response, running), and may or may not be performing the same tasks. The system typically must dedicate system resources to manage the session of each connected user. Caution: has been called a “dangerously misleading statistic”.

***Configuration*** The state or settings of a system and its environment, to ensure the system runs in the desired manner.

***Configuration Control*** The control of changes to the configuration settings in software or its support environment. See change control, configuration management.

# *Appendix A: Basic Definitions and Concepts*

***Configuration Item*** The software component, support hardware, document or data which is being controlled.
***Configuration Management*** The process of managing the different states which a system and its environment can be in. See version control.
***Configuration Test*** Configuration testing is testing the product under a different number of hardware or software configurations. Hardware and software can sometimes have an unlimited number of configurable items and this would be too much to test, so it is advisable that the product be tested with, at least, the minimum, the maximum configuration and a few different configurations in between. Verifies performance impacts from changes in software components, hardware, processor speed, memory size, etc. (1) Testing of an application or a technical environment (such as a DBMS) for compatibility across different configurations of a system, e.g., different client/server network configurations. (2) Use of information about the internal state of the environment to check the stability of the system in different configurations. (3) Checks that a system works acceptably in the various different configurations to which it can be set.
***Conformance Test*** The process of testing that an implementation conforms to the specification on which it is based. Usually applied to testing conformance to a formal standard.
***Connectivity Test*** See interoperability test.
***Contention Test*** Verifies the target-of-test can acceptably handle multiple actor demands on the same resource (data records, memory, etc.).
***Context-Driven Testing*** The context-driven school of software testing rejects the concept of best practices which are not context-specific, and the “one approach fits all” tone of some test standards and processes.
***Context Switch*** The process of switching from one task to another in a multitasking operating system.
***Conversion Test*** (1) Checks the integrity of a system conversion, change in environment or database migration. (2) Testing of programs or procedures used to convert data from existing systems for use in replacement systems.
***Corner Case*** A combination of extreme but valid conditions.
***Corrective Action*** An action taken to prevent or recover from a problem.
***COTS*** Commercial off-the-shelf software.
***Coverage*** Percentage of testable conditions that actually are tested. The measure of testing completeness for a particular testing strategy.
*Coverage Analyzer* Like path analyzers, coverage analyzers and profilers work with the source code and are language dependent.
***Coverage Tracking*** Tracking the conditions which are being covered during the execution of a test suite. Using tools, automatically track and report which conditions (i.e., paths or branches) have actually been tested. Helps understand the completeness of testing.
***CPU*** Central processing unit -- the processor or brain of a computer; the part of the computer which executes instructions.
***Crash*** An abnormal system termination of action.
***Critical Section*** A sequence of software instructions which must be executed in sequence and without interruption. See also race condition.
***Cross-Compiler*** A compiler which operates on a different hardware platform or operating system than the one for which it produces machine-level processor-specific (object) code.
***Cross-Functional Analysis*** While classic functional analysis focuses on testing the individual functions of a system, cross-function analysis provides an additional test of the relationships or interdependencies among the functions. This technique requires sufficient understanding and knowledge

of the system to be able to identify the relationship among the functions. It is also called feature interaction testing.

***Cross-Platform Test*** Testing to ensure an application system works the same way on different platforms (e.g., Windows and Unix), or alternatively that components of an overall system work together across different platforms.

***Customizability***. This is the built-in flexibility to customize the system for different users. It is similar to configurability and maintainability.

***Cyclomatic Complexity*** A measure of the logical complexity of an algorithm, used in white-box testing, it is the count of basis paths in a software module. Used as a measure of the difficulty and likely bugginess of a software component or algorithm. Also called McCabe's measure.

***Daemon*** A software program which performs a specific function on a computer system.

***Data Center Operations Test*** The purpose of these tests is to ensure that the data center and/or network administrators can safely proceed with the installation process, without inadvertently affecting other operations. Usually not performed by the users as part of their acceptance test, but performed in parallel by data center operations and network professionals.

Data conversion test. Ensuring that data has been converted correctly (for data base conversion), or that the system still performs appropriately after migration to a new technical environment (e.g., upgrade to a new release of the network software on which an application depends).

Data dictionary. A list of the definitions of the data items, together with descriptions of their attributes, which are used by a system or stored in a database. The database contains the definitions of all data items defined during analysis.

Data element. See data item.

Data field. See data item.

Data flow anomaly detection (DFAD). Check for anomalies in data use which may indicate a programming error: (a) a variable is referenced without previously having been assigned a value, or (b) a variable is assigned a value but not subsequently referenced.

Data flow diagram (DFD). A visual modeling notation which shows the flow of data and the series of steps which manipulate data with in a process or a system, and represents a functional decomposition of the system.

Data integrity. Can the accuracy of stored data be sustained?

Data item. A specific, atomic piece of data. Also called a data element, a data field or an attribute.

Data modeling. The method of identifying data needs, defining the data relationships, organizing and structuring the data, and defining the data attributes, so that it can be implemented as a database. See entity, relationship, attribute.

Database. Organized, structured repository of data.

Database demographics. Profile of the occurrence and representation of data in data bases (e.g., the breakdown of foreign orders by country and type of customer) and data usage patterns, in order to determine what types of data values (e.g., what types of customers) to test.

*Data Base Integrity Manager* These tools check and maintain the test data base(s). They check the business rules and referential integrity (e.g., no order may exist without being related to a customer), protect the test data, and correct test data that has been corrupted by SUT errors.

***Database Integrity Test*** Checking the integrity of data field values stored in a database, by sampling and comparing the sampled data to known correct reference data. Can include testing the data field integrity, business policy rules, access controls, and reliability of the processes which maintain or refresh a data base.

Data-driven testing. Testing in which the action of a test case is parameterized by externally defined data values, maintained as a file or spreadsheet. A common technique in automated testing.

# *Appendix A: Basic Definitions and Concepts*

***Deadlock*** A situation in which a set of interdependent tasks is blocked, with each one waiting for an action by another one of interdependent tasks. A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function. Here is the example: Program 1 requests resource A and receives it. Program 2 requests resource B and receives it. Program 1 requests resource B and is queued up, pending the release of B. Program 2 requests resource A and is queued up, pending the release of A. Now neither program can proceed until the other program releases a resource. Deadlock often arises from adding synchronization mechanisms to avoid race conditions.

***Deadlock Test*** This type of testing attempts to stress a system by locking a database through transactions which interfere with each other.

***De-allocation*** The process of freeing memory when an application no longer needs it. xecution.

Debugging. The act of diagnosing a problem and finding and correcting its cause (an error or defect). The process of finding and removing the causes of software failures.

Decision table test. A technique to develop test cases from decision tables. Develop a decision table showing the decision criteria of a process, and the actions to be taken for each viable combination of these criteria. Test each distinct column in the decision table.

Defect. The specific cause of a failure. Synonym: bug, fault.

Defect propagation. The tendency of a defect, left unfixed, to cause more defects.

*Defect Tracking Tool* Used to capture, track and analyze defects, and maintain a defect repository. Also commonly called a problem reporting tool.

***Degraded Mode of Operation Test*** Systems are designed to use a given set of resources, such as hardware, networks and databases. Their users expect many systems to provide ongoing service, even at reduced rates of performance and capacity, when not all the resources are working (e.g., a database is unavailable). The purpose of degraded mode testing is to determine whether the system can still provide the reduced level of service as expected. An example of a degraded mode test is to deliberately power down an application server, in a server cluster with redundant application servers, and attempt to continue normal operation Test to ensure that a system continues to operate even after portions of the system become inoperable, continuing to provide either the full, normal or less-than-full capabilities of the system. Examples, testing the back-up dial-up mode when the direct line connection is down, testing capabilities in a multi-server network when one of the servers is disconnected. Unplug the machine or network when in the middle of executing the application. Press other keys or attempt to do another operation while executing the application. Decrease the amount of memory or system resources available for the application.

***Dependability*** The term “dependability” is often used to summarize the degree to which we can rely on a system. This term is broader than just reliability and recoverability – it encompasses security, safety and data integrity too.

***Design*** (1) The activity which translates the system requirements into a blueprint of the proposed technical solution, which is the system design. (2) The result of the design process – the specification of the technical solution to meet system requirements. See: architecture, design specification.

***Design for Testability*** As systems become more complex, it becomes more difficult and eventually impossible to test them adequately unless they have been specifically designed to be testable. Much of a system's behavior may be hidden and not directly observable from the outside, which severely limits the effectiveness of non-invasive black-box testing. For example, an internal buffer overflow may be extremely difficult to observe in testing or in live operation, unless a capability has deliberately been designed into the system to provide this information. To be testable, a system has to be (a) observable and (b) controllable. A system is relatively easy to observe if the outputs from that system are dependent only on the inputs, regardless of the internal state of the system or the state of its supporting

infrastructure. But is not easy to test without monitoring the internal behavior of the system, if the outputs are dependent not just on the inputs but also on hidden, transient internal states of the system. Designing systems for testability is often not done very well. In small, simple systems, the system architecture is fairly obvious to the test professionals, and there is a ready availability of access points to observe the internal states of the system. It is in large, complex systems that designing for testability becomes more important and also, unfortunately, much more difficult. Usually the main problem is one of communications. Designing for testability requires a solid gray-box understanding of the system.

Desk checking. An individual programmer's review of source code, before or after compilation, usually performed by the author himself or herself.

Device driver. A software module that hides the details of a particular peripheral and provides a high-level programming interface to it

DFD. See data flow diagram.

*Diagnostic Tool* These tools monitor the SUT, to gather data which is used later to pinpoint, isolate, debug and fix problems. They may place invasive probes into the SUT, or provide information about the environment. For example, tools can provide information to analyze the memory contents of the hardware on which the SUT runs (a memory dump).

Digital signal processor. A specialized processor which is designed for discrete-time signal processing. DSPs usually support instructions to perform signal processing, such as in telecommunications transmission and reception.

***Dimensional Analysis*** This is a simple way to find errors in calculations, by checking that the units of measure (the dimensions) are consistent on both sides of an equation, and avoid comparing apples to oranges. Consider for example the question of how many miles a car travels in a quarter-hour if it is traveling at a speed of 60 miles per hour – the answer is 15 miles. We derive the answer by using this equation: [ Distance traveled = Speed multiplied by Time ]. Speed is measured in miles per hour and Time is measured in hours, so multiplying them together effectively means that the dimension of the left side of the equation is miles (miles/hour multiplied by hours). This should be (and is) consistent with the dimension of the right side of the equation (also miles).

***Disaster Recovery Test*** Simulating the occurrence of disasters, for the purpose of ensuring the system's recovery processes are capable. This type of testing uses the disaster scenarios which were identified in the organization's disaster recovery plans as a source of test cases. I'll illustrate this point with an example of a system failure. In what was considered a major crisis, the Nasdaq stock market halted trading on a busy Friday in 2001. An employee of WorldCom, which provides communications services to the stock exchange, had inadvertently forced Nasdaq's communications network to shut down. (WorldCom later said that testing a new system being developed for Nasdaq had caused the service interruption. With hindsight, a busy Friday was not a very mart time to run the system test.) Nasdaq was able to restore service fairly quickly, but a secondary problem blocked its stockbroker clients from using the system for several more hours. The outage had disconnected all of Nasdaq's users from their network. When these users attempted to log back in after the network administrators had resolved the problem -- and all at approximately the same time -- the system's logging process was unable to cope with the huge surge of demand. This scenario is similar to the idea, which testers have often tossed about, of conducting a stress test by having 1,000 people hit the Enter keys on their keyboards simultaneously (or simulating the same activity with testing tools). So what's this got to do with the price of fish? The connection is as follows. Testers often have difficulty identifying stress test cases. These testers may not be very close to the day-by-day system operation and so have difficulty visualizing how the system might fail. In addition, the test cases which the testers do identify can seem highly implausible. Observers may guffaw at the thought of 1,000 people hitting the Enter keys simultaneously: they tell the testers to "get real", "give us a break" and "do something worthwhile". In the case of Nasdaq, their disaster recovery plan had identified the possibility of tens of thousands of clients trying to log on together

after a system failure. But the testers had not referenced this document, and so this particular scenario had never been the subject of a stress test.
Disclaimer. A statement which warns or risks and limits liability.
***DNS*** Domain Name Server, a device that provides Internet addresses.
DOD. U.S. Department of Defense, a major developer and user of software
DSP. Digital signal processing.
Dumb monkey. Automated test tool which has little or no knowledge of the functioning of the AUT. Dumb monkeys do things like mindlessly clicking on buttons to see what happens. Contrast with a smart monkey.
Duration See session duration; test duration.
***Duration or Endurance Test*** This type of testing places a load on the system for an extended period of time, usually for a few days, with the purpose of detecting slow-to-appear defects such as memory leaks or buffer overflows. Duration testing may also be used to measure the system availability, in terms of the percentage of the time it is functioning, and reliability of the system, in terms of MTBF (mean time between failure). People also call this type of testing long-sequence testing, burn-in testing and soak testing. With the fast processing speeds and the parallel processing which are available today, more testing can be done more rapidly, and extended test durations arguably have become less important. Especially because an entire project may have to wait for days while the duration test runs, it is a good idea to review the bug reports to see what new information, if any, the second hour's worth of testing adds beyond the first hour, the third hour beyond the second, and so on. In addition, the field of software reliability engineering does not require inordinately long test times in order to evaluate system reliability.
Dynamic test. One that requires actual execution on computer hardware, networks, etc., as opposed to a static test. Testing software through executing it. See also Static Testing.
E2E. End-to-end testing
ECO. See engineering change order.
***Embedded System*** A system which is embedded in a device and does not have a conventional user interface, such as Windows-based GUI. An example is the anti-lock braking system in a automobile. Embedded systems combine hardware, software and possibly electro-mechanical devices to perform a particular function,
Embedded system test. The testing of embedded systems, i.e., those which operate directly on some low-level device such as a video chip or a set-top box.
Emulator. A tool, which usually combines hardware and software, which emulates (stands in place of) a system for development, testing and debugging purposes. For example, a WAN emulator mimics a wide area network when a real WAN is not available for testing, and an in-circuit emulator (ICE) emulates a processor (CPU). See simulator. A device, computer program, or system that accepts the same inputs and produces the same outputs as a given system.
***End to End Response Time*** Response time measured from the end-user perspective. For example, the time between when a user presses the "login" button and when the subsequent page is fully loaded. AKA perceived response time.
End-to-End test. Testing a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.
***Endurance Test*** Checks for memory leaks or other problems that may occur with prolonged execution.
Engineering change order (ECO). A specification of the technical aspects of a change.
Entry criteria. Conditions to be met before a specific activity begins. Entrance criteria are most effective when the are aligned with the significant risks associated with starting the activity when the criteria have not been met.
Environment. (1) The supporting infrastructure within which a system operates. (2) The related systems

with which a system interfaces.

***Environmental Test*** Any test that does not primarily concentrate on functional or feature correctness, but instead concentrates on either: (1) How the development and maintenance process affects the testing process (e.g., modifications vs. new development, software packages vs. custom-built solutions), and (2) How the system will operate in its real-world environment (e.g., performance, usability, security controls, compatibility across different configurations or platforms). This method considers the environment within which the system is being used. For example, a system may be installed on a factory floor where there is heavy vibration and industrial chemicals in the air. Another system may be used on a plane at an altitude of 50,000 feet, where electromagnetic radiation can arbitrarily change bit settings in magnetic media.

***Equivalence Class*** A set of input conditions or data values which produce the same result or an equivalent result. If one test case within the equivalence class works correctly, it can be assumed the rest will also work correctly, so that they do not all need to be tested. A portion of a component's input or output domains for which the component's behavior is assumed to be the same from the component's specification.

Equivalence partitioning. A testing method where input data values (or other items) are partitioned and then grouped together into equivalence classes. A variation the domain/range test approach for cases where an input can cause multiple internal behaviors. Divide each domain into partitions (a set of ranges, within each of which the system behavior should be the same). Test one representative value within each partition, and assume that its result correctly predicts the results for all other values in that same partition.

Equivalence. Commonality, similarity.

***Equivalence Test*** A test which uses only a small sample of all possible test conditions, but ones which are chosen intelligently so as to uncover almost as many defects as an exhaustive test would have uncovered.

Error. Specific cause of a failure. A defect is also referred to as an error or colloquially as a bug. Most defects lurk in products indefinitely and are ‘benign’ (i.e., are never exercised or never cause a noticeable failure. Unacceptable behavior; a lack of conformance with what the users can reasonably expect. Synonym: bug; defect: anomaly.

Error guessing. A test case identification technique where the approach is to blue-sky brainstorm about where and how errors are likely to occur and develop corresponding test cases. Can be creative and cannily focused when based on shrewd intuition, but may not lead to a reliable test coverage.

Error message. A message produced by a system to notify a user, operator or support person of a problem encountered in system operation.

***Error Processing Test*** Some products, especially in a client server environment, interact with other applications, systems or other platforms (Client to middle tier to back-end or server) and during this process the product may traverse through many different paths. This can be an exhaustive error-processing test. It is recommended that in this case the tester should first concentrate on their assigned application error processing first and then coordinate with other projects to ensure that the error processing related to their respective areas is conducted. However, the test engineer should not just rely on other projects to test errors that may relate to the assigned application when it is communicating with other parts of the system. Also the test engineer should read the error messages returned and determine if the error makes sense to the audience it is intended for. For example, error messages are different for users versus system programmers.

***Error Rate Measurement*** Let’s say that the response to a database query happens in 0.1 second, but this response says: “Database not available”. Or that a Web service can handle 10,000 users simultaneously, but 500 of these users receive error messages. Fast response time and high throughput are irrelevant if the user can’t do his or her job. Since this type of testing counts the incidence of errors or failures, the first thing we need is a catalog of errors. I’ll provide a list of the types of failure later, in the section entitled “Modes of Failure”. If you glance ahead at that list, though, you will quickly see it is not

particularly suitable for our purpose here. The list contains items relevant to system and network administrators, such as “race conditions: timing out of sync.”, “memory leaks”, “page locking”, and “processor saturation”, but which are meaningless to the end users. In other words, we need a user-centric list of errors. We need to distinguish between symptoms of failure and causes of failure. Here we will be focusing almost exclusively on the symptoms, not the underlying causes, and only those symptoms which and meaningful to the end users see. In addition, most of the items on the user error list are not catastrophic errors (“dark screen”, “dead keyboard”), but niggly and momentary annoyances to which the system administrators may be oblivious.

Error seeding. Deliberate insertion of errors to assess test effectiveness, based on which inserted errors are found. See: bebugging, software fault injection.

Event partitioning. In event-driven testing, deliberately stabilize or freeze selected variables, and isolate events from each other (e.g., by deliberately disconnecting selected input sensors), in order to provide a simpler set of test cases. Event partitioning usually occurs prior to more complex tests of multiple simultaneous events, if these occur in the system being tested.

***Event-Driven Test*** (1) Testing event-driven processes, such as unpredictable sequences of interrupts from input devices or sensors, or sequences of mouse clicks in a GUI environment. (2) A technique to develop test cases based on events. In situations where events occur concurrently or in sequences that cannot be easily predicted, test each event individually first. Then, test an appropriate sample of representative sequences or simultaneous combinations of events.

Exception handling. A test case identification technique where the approach is to identify the error messages and exception handling processes, together with the conditions which should trigger them. Test each set of error conditions.

Executable. (1) Ready for operation. (2) Short-hand for a executable file.

Exhaustive test One which exercises every possible condition. Alternatively, a test of every possible input and every possible set of initial conditions. Testing which covers all combinations of input values and preconditions for an element of the software under test.

Exit criteria. Conditions which should be met prior to or when completing a specific activity.

***Expected Peak*** The most likely value or height of the peak. In a situation where the peak can vary, e.g., based on the length of the time duration which is being observed, the expected value is the modal (most frequently observed) value.

***Exploratory Test*** This is a "learn as you go" technique with quick feedback loops, used where a system is not well understood and not well documented, and where there is no time for extensive data gathering. The idea is to perform an iterative, evolving series of short test cycles: test the system on an exploratory basis, learn about the system through the testing, decide based on the test results where next to focus the test effort, and then move on to the next iteration of testing. No extensive, formal test plan or test cases are developed prior to starting the testing.

Exposure. Importance of a risk, calculated using the equation: [ (Cost of Defect) multiplied by (Probability of Defect being triggered when a condition is encountered) multiplied by Frequency of Execution of the Condition)]

Extreme programming (XP). Software development methodology developed by Kent Beck and others. It emphasizes strong unit testing, incremental development, evolving design and a light application of controls and documentation.

***Failure*** (1) Deviation of a system or component from specified or expected behavior. This definition assumes that the functional specification in itself is adequate or the expectation of the system behavior is reasonable. (2) Any incident with negative consequences, where a system or component deviates from its expected behavior. Synonym: anomaly, incident, malfunction.

Failure modes and effects analysis (FMEA). A technique to develop test cases based on failure analysis. Analyze the block diagram or similar schematic that shows the system design architecture or structure, to

identify likely points of failures (e.g., critical interfaces among components). Modify the design before the start of coding to address these potential failure modes (e.g., add additional controls). Use the FMEA results also as a source of test cases.

Familiarization testing. See exploratory testing.

***Fault*** The underlying cause of a failure.

***Fault-Based Test*** A technique for testing computer software using a test method and test data to demonstrate the absence or existence of a set of pre-defined faults. For example, to demonstrate that the software correctly handles a divide by zero fault, the test data would include zero.

***Fault-Tolerant Test*** Tests the ability of a system to continue operating despite faults, though possibly in degraded mode, despite the occurrence of faults. This is similar to error detection and recovery testing, degraded mode testing and robustness testing.

Fault tree analysis. Fault tree analysis identifies the faults (hazards) that can occur in a system and uses a cause-effect graph or a process flow model to trace back to the events and causes which trigger each possible occurrence of the fault. It is similar to FMEA and hazard analysis.

FCS. First customer ship: it means either the release date and/or first released version of the product.

Feasibility study. Analysis of the business justification and technical feasibility of a project.

Feature. (1) A capability provided by a system to it users. (2) Unpleasant experience, as used sarcastically in the statement: “It’s a feature”. From the adage "It's not a bug, it's a feature."

***Feature Interaction / Interference Test*** This type of testing attempts to stress a system by having features, processes or threats interfere with each other. Suppose that a system contains two features, A and B. In the feature testing, we run a set of test cases to exercise feature A, and another set for feature B. These features can interact and possibly interfere each other, for example, by both being able to simultaneously access and update the same records in a common database. In this situation, the feature testing usually includes feature interaction testing, but only to a limited degree and for the simplest cases. For example, in a manual test two different testers, working from two workstations, attempt to update the same database record at the same time. Usually the feature test team cannot easily try more complex combinations of many concurrent activities. A load or stress test, by contrast, by its nature usually incorporates multiple concurrent demands on the system we are testing. In the feature interaction variation of a load test, we deliberately engineer the test work load to include “nasty”, complicated and interacting mixes of demands.

Field test. A test which addresses these questions: Does the system perform adequately in a pilot, beta, trial production or parallel environment? (Parts of these tests may be a customer or user responsibility.) Pilot and field and beta testing are similar.

FIFO First in, first out. A way of managing a queue.

Filtering. Sorting through large volumes of information and presenting to the user those which are likely to satisfy his or her information requirement.

Firmware test. The testing of firmware (low-level software that is usually embedded and pre-loaded into ROM or read-only memory).

Fix. A modification to a system to resolve a problem.

FP. See function point.

***Framework*** In automated testing, the infrastructure to support the testing tools.

Front-end. The front-end of a project includes all project activities that occur before the coding begins.

Function point. A measure of the software size, based on its functions and derived from the inputs, outputs, data storage and complexity of that software.

Function. A capability of a system. See: feature.

Functional analysis. Analyzing the functions or features of a system to derive test cases for those functions, and ensures each function of the system is adequately tested. Traditional functional testing is not concerned with performance, usability or stress-related activities, such as running multiple functions

in parallel. This method is also called feature analysis and function-output-input (FOI) analysis.
Functional decomposition. (1) The development of a functional hierarchy for a software design. (2) The work breakdown structure for a project. (3) A technique used during planning, analysis and design; creates a functional hierarchy for the software.
Functional specification. A document that describes in detail the characteristics of the product with regard to its intended features.
Functional test. See also black box test.
Functionality. The collected set of functions for a system.
***Garbage Collection*** The collection and re-setting (cleansing) of memory fragments as they are freed by de-allocation, ready to be allocated for the next use.
Gateway. See quality gateway
General-purpose computer. A computer which does not have a specific, dedicated purpose but can be adapted by software for diverse purposes, for example, a personal computer. Contrast with an embedded system.
Glass Box Testing. A synonym for White Box Testing.
Go/no-go decision. A project review point when the project manager, marketing or the customer decides whether a project should proceed.
Good enough software. Software which meets the users' needs and acceptance criteria, but is not tested and debugged beyond that level.
***Goodput*** Measured in bits-per-second, shows useful application data successfully processed by the receiver. It measures effective or useful throughput and includes only application data—not packet, protocol, or media headers.
Gorilla Testing. Testing one particular module or set of features, fairly heavily.
Grammar-based test. See BNF, syntax-driven test.
Gray box test. An intermediate level of testing, between white and black, where a sufficient design-level view of the internal structure is available to test component interfaces and sub-assemblies. See integration test
GUI test. Testing the front-end user interfaces to applications which use GUI (Graphic user interfaces.
GUI. Graphical user interface.
Guideline. Provides guidance without necessarily being a mandatory standard.
***Hammer Test*** These tests have little or no resemblance to real-world distributions and user communities. A typical test takes all the existing load generation scripts and methods, eliminates user think times and increases load until failure occurs. These tests are designed to make the system fail. Once failures occur, mitigation strategies can be determined. These tests are generally only executed after several rounds of tuning. There are no pass/fail criteria for hammer tests as the intent is to find the break points and develop risk mitigation strategies from those break points. See also hot spot test.
Hang. System freeze.
Hardware. The processors, memory and peripheral devices on which software runs.
Hardware component. A piece of hardware, such as a transistor or a router.
Hardware/software co-development. A development approach where the software is being developed before the hardware is available, and vice versa.
Hazard or threat analysis. A technique to develop test cases based on identifying what could go wrong with the system (the hazards or threats). Determine what the appropriate response should be to each hazard or threat (the system's expected behavior). Use this analysis as the basis for test cases, to validate that the system responds to hazards in an acceptable way.
***Hazard or Threat Identification*** This method, which is similar to risk-based testing, considers the dangers and obstructions associated with using a system.
Hazard. See risk.

# *Appendix A: Basic Definitions and Concepts*

Heap. An area of memory which is used for dynamic memory allocation. See allocation.
***Heartbeat*** Coordinating status message among servers, used in server fail-over.
Help desk. User support facility, similar to a technical support group but usually internal to an organization.
Help test. See on-line help test.
High-level architecture. “Big picture” overview of the system structure.
High-level design. See high-level architecture.
High-level integration. Integration of large-scale or major components and subsystems; conceptual view without detailed interface data formats and definitions.
High-level language. A programming language, such as C or Java, which is used by software engineers to write software which is then transformed by a compiler into machine-readable code (also called object code or executable files). HLLs are usually portable, i.e., processor-independent.
High-order test. Black-box test conducted once the software has been integrated.
***Hit or Page Hit*** The retrieval of any item from a web site, such as a graphic file. Typically more than one hit is required to retrieve an entire page.
HLL. See high-level language.
Home page. The primary point of access to a Web site, which provides links to the related pages in the site.
Host. (1) The platform on which software operates, or the target platform on which it will operate. (2) A mainframe computer.
***Hot Spot Test*** This is a variation of stress testing, where the demand on the system is heavily focused on a specific, limited portion of the system, in order to detect if it is a weak point. We employ it in areas which we suspected are vulnerable to stress, for which, the weak links I the chain which are likely to break first under load..
***Hourly Users or Visitors*** The total number of individual users who have an open active session during the span of an hour.
***Hourly Visits or Sessions*** The total number of unique sessions during the span of an hour. Since each user or visitor must have a minimum of one session, but could have more than one within the span of an hour (by logging off and then on again), the hourly visits must be equal to or greater than the hourly visitors.
HTML. HyperText Markup Language, a common way of inserting simple formatting instructions into a web page.
HTTP. HyperText Transfer Protocol, a protocol widely used on the Internet.
HTTPS. HyperText Transfer Protocol, Secure. This is a variant of HTTP for secure transactions.
Hyperlink. Connector between Web pages or documents.
HyperText Markup Language. A set of standard tags and codes used to format hypertext documents.
HyperText Transfer Protocol. Protocol for the Internet, whereby a client and server communicate during a hyperlink.
***“Ilities***” Non-functional requirements used to assess the quality of a system, such as usability or reliability.
I/O device. A device (usually a hardware component) which interfaces between a processor, its peripherals such as printers and the external environment.
I/O. Input/output: the interfaces between a system and the other related systems with which it interacts.
ICE. See in-circuit emulator.
IEEE. Institute of Electrical and Electronics Engineers.
***IIT*** See initial impact assessment or infrastructure impact assessment.
Implementation. (1) The process of delivering a system. (2) The delivered system.

In-circuit emulator. A tool which emulates a processor (CPU) for development, testing and debugging purposes.

***Increasing Work Load*** In a load test, an increasing work load is a type of work load used to find the limit of a Web application's work capacity. Virtual users are added to the test load until the application is stressed to the failure point.

Independent test group (ITG). A group of people whose primary responsibility is software testing,

***Independent Verification and Validation (IV&V)*** A phased approach to testing, primarily used in the military and by defense contractors, which combines both validation and verification.

Independent verification and validation (IV&V). A phased approach to testing, primarily used in the military and by defense contractors, which combines both validation and verification. An example of these terms follows. Validation: the contractor validates that the system meets specifications. This activity happens at the contractor's facilities. Verification: the client (usually military or government) verifies that the system meets the client's requirements. This second activity usually happens at the client's facilities.

Information services. Common name for the internal roup with provides computer services to an organization.

Information technology. Common name for the internal roup with provides computer services to an organization. See information services.

***Infrastructure Impact Assessment*** Assesses the impact of a change on an existing infrastructure which is supporting a mix of work load demands. This testing focuses not so much on the immediate application being changed, or the new application which is being introduced into the environment, or a change in the demand patterns within one application, but on its side effects on the other uses of the infrastructure. The issues are the capacity and the utilization of resources in that infrastructure. This is also called an environmental assessment.

***Initial Impact Assessment*** A quick, approximate form of an infrastructure impact assessment.

Input domain test. For each input, identify each domain over which the system behavior should be the same. Test one representative value within this range, and assume that its pass/fail result correctly predicts the results for all other values in the range.

Insider. Member of the inner core team which develops or maintains a system.

Inspection. A visual examination to detect errors, violations of development standards, and other problems. A group review and quality improvement process for written material. It consists of two aspects; product (the document itself) improvement and process improvement (both document production and inspection).

Installability. Can the system be quickly and easily installed on a variety of platforms by a variety of users? Is the migration or conversion from existing systems, databases and business processes well understood and reliable?

***Installability Test*** The successful installation and execution of the product across different software and hardware configurations is an important part of testing the product. This type of test can be combined with configuration and compatibility testing. Sometimes the product may have a set of installation diskettes or installation procedures that require testing. The installation process should be documented in the early stages of the development process and put into the configuration management plan. This will make the install process easier for the implementation phase of the development process.

Installation. (1) The process of installing a system. (2) The installed system.

***Installation Process Test*** This tests the system installation process, which is important to vendors and to information services (IS) organizations who plan to install a system in many field sites. (This is not the same as software package installation testing, which also typically includes a functionality check-out and confirmation of a valid data conversion.)

***Installation Test*** (1) Checks the system, network or database installation process. May be combined

with the conversion test on smaller projects. (2) A test which addresses this question: Does the system installation process work correctly. Confirms that the application under test recovers from expected or unexpected events without loss of data or functionality. Events can include shortage of disk space, unexpected loss of communication, or power out conditions.
Instruction pointer. A register in the processor which contains the address of the next instruction to be executed.
Instruction. A statement or directive or line of code.
Instrumentation. The insertion of features directly into a product to aid in testing and debugging, such as diagnostic traces, defensive re-editing of passed-along inputs. May be temporary or permanent;
Integer. A whole number without fractions or digits after the decimal point, such as 0, 27, and -753.
***Integration Test*** This system may interface with one or more other external products, or be a component of a larger application. In this case, the product must be tested against the other interfaces it's communicating with or through. For instance, in a client server environment there may be interdependencies between one product and another that communicates through some other interface via the Internet. Also in a multi-tiered environment the test team must take into consideration the interfaces between the client, middle tier and the back-end. Each piece of the n-tiered environment is integrated separately, tested separately and then tested again when all of the different pieces are integrated to make a complete operational system. (1) A test of a combination or sub-assembly of selected components in an overall system. Integration testing usually is incremental, in that successively larger and more complex combinations of components are tested in sequence, proceeding from the unit level (0% integration) to eventually the full system test (100% integration). (2) A test which exercises a combination or sub-assembly of selected components which eventually will be part of an overall system. (3) The term integration test can also be used in a different way, to describe large-scale systems interface or interoperability testing. In this use of the term, integration testing comes after the traditional systems test. (4) Testing of combined parts of an application to determine if they work together correctly. Usually is performed after unit testing. This type of testing is especially relevant to component-based and distributed systems.
*Interface Simulator* In live operation, the SUT usually interfaces with other systems. These other systems may not be present in the test lab, because of feasibility, availability and expense, or because the SUT is deliberately being tested in isolation. The interface simulators replace and crudely mimic the behavior of the other systems.
Interface test. Test an interface by passing parameters (data fields) across the interface, and checking that the correct response occurs.
International Standards Organization. European-based standards setting group.
Internationalization test. Testing that national-market versions of a product work in their target markets (e.g., in France or Japan). Also called a localization test.
Internationalization. The preparation of software and hardware for international markets, usually with multilingual capabilities. See localization.
Internet Protocol (IP). The network layer for the TCP/IP protocol in the Internet. It provides packet disassembly, routing, and re-assembly services.
***Interoperability Test*** Identify the interfaces, data feeds and controls among components, subsystems or complete systems. Use this as the basis for testing these interfaces. In different variations, this is also called integration testing or system interface testing. Interoperability testing is only as effective as the testers' ability to see and understand the interfaces. Also called a connectivity test.
Interoperability test. Identify the interfaces, data feeds and controls among components, subsystems or complete systems. Use this as the basis for testing these interfaces. In different variations, this is also called integration testing or system interface testing. Interoperability testing is only as effective as the testers' ability to see and understand the interfaces. Also called a connectivity test.

Interoperability. Does the system interface correctly with the other systems it is expected to work with? Interoperability is the ability of a system to communicate, connect and or interface with other systems.
Interrupt service routine. The software component which is initiated when a particular interrupt occurs.
***Interrupt*** (1) The temporary suspension of a running software process or task to give resources to another higher-priority task. (2) A signal from a software component or a hardware device such as a peripheral. When an interrupt occurs, the current state of the saved for future restoration of processing after the interrupt has been taken care of, and an interrupt service routine (ISR) is executed. When the interrupt service routine exits, control is returned to the software component which previously was executing.
Inter-task communication. A mechanism used by tasks and interrupt service routines to share information and synchronize their access to shared resources. The most common building blocks of intertask communication are semaphores and mutexes.
Intranet. An internal, private network which provides services like the Internet but is not necessarily connected to the public Internet.
Invasive test. A test in which the object being tested has been modified by the introduction of monitors, internal probe points and sensors required for the test process. Software generally needs to be re-compiled or re-built to incorporate these probes. Synonym: intrusive testing. See: instrumentation.
IP. See Internet Protocol.
IPL. Initial program load. See start-up code.
IS. Information services.
ISO. International Standards Organization.
***Isolation Test*** An isolation test is a type of load test which focuses on a particular point in an application by continually executing a specific set of transactions. Isolation tests help in pinpointing a problem once one has been identified to exist in the application. An isolation test is a type of load test used to focus in on a particular problem in a Web application. It is used once a number of load tests have identified that a problem actually exists in the application. Isolation tests often execute specific sets of transactions over and over again in order to precisely identify a problem. These tests can help in pinpointing, for example, which set of requests caused a Web server to send back an error message or caused a deadlock on the database server.i
ISR. See interrupt service routine.
ISTQB. International certification for software testers.
IT. Information technology.
Iteration. A single pass through a group of instructions. Most software contains loops of instructions that are executed over and over again. The computer *iterates* through the loop, which means that it repeatedly executes the loop.
Iterative development. Method of software development where the requirements and design are not frozen but evolve as the coding is being done.
Iterative or spiral life cycle test. Testing in a environment where the specifications and product are expected to continue to evolve indefinitely, and where there is never a frozen, final or documented specification. Also called a RAD (rapid application development) test or a prototype test.
IV&V. Independent Verification and Validation.
***JAD*** See joint application development.
JIT. See just-in-time software.
Joint application development (JAD). A technique for requirements definition and system design which emphasizes user involvement in joint sessions with the technical developers.
JPEG. Joint Photographic Experts Group format: a standard for graphics files.
Just-in-time (JIT) software. Software developed, acquired or modified just as needed, in order to minimize obsolescence.

# *Appendix A: Basic Definitions and Concepts*

***Kaizen***. Continual process improvement.
Kernel. The core of a software architecture. In a multitasking operating system, for example, the kernel contains the scheduler and context-switching algorithm.
KLOC. Thousands of lines of code.
Knowledge Management. The systematic management and use of the knowledge in an organization; the leveraging of collective wisdom to increase responsiveness and innovation."
KPA. Key process area. See: CMM.
LAN. Local area network.
Latency A delay or the possibility of a delay; a wait state.
***Level of confidence in data***. The confidence ranges from low to high:

- Very Lo: there is a 50% or better chance that the real data value does not lie within 30% of the measured or reported data value.
- Lo: there is a 50% or better chance that the real data lies within 15% of the measured or reported data value.
- Mid: there is a 90% or better chance that the real data value lies within 15% of the measured or reported data value.
- Hi: there is a 90% or better chance that the real data value lies within 5% of the measured or reported data value.
- Very Hi: there is a 95% or better chance that the real data value lies within 5% of the measured or reported data value.

***Limit Test*** Identify the limits of the system, and test what happens when the system is pushed to and beyond these limits. For example: a counter overflows (i.e., an attempt is made to increase its value beyond the maximum possible value, based on its size). See also stress test.
Line of code. Source instruction or statement in a programming language. Also referred to as an LOC.
***Linear Projection*** Involves collecting past data, extending or implying trends through the use of scatter plots and regression lines, and comparing the trend line with the current capacity of the system. Although linear projection is used quite frequently to make assumptions about future behavior, the performance of computer systems is far from linear. Therefore, any linear relationship determined between two system components should only be used to understand the current (or past) behavior. Linear projection is best used when you want a first approximation to a very simple model.
Line-of-code metric. Measure of quality or productivity which is normalized based on the source code (i.e., multiplied or divided by the number of lines of code).
Linker. A software tool which runs after the source code has been compiled, and transforms the object files into relocatable software (i.e., software which can bed moved from location to location in memory without loosing the linkages among the files).
***Little's Law*** xxx
***Live Change Test*** There are many systems which must keep running, no matter what. One example is an aircraft flying over the ocean. What happens when an emergency fix or routine maintenance must be done without taking the system down? This type of testing assesses the ability to make live modifications to the system without interrupting service.
***Live Data Test*** Extract a sizeable volume (a copy) of existing live production data, modify it as needed for the test, execute it and see what transpires.
Live data test. Extract a sizeable volume (a copy) of existing live production data, modify it as needed for the test, execute it and see what transpires.
Live. Active, working, operational.
***Load*** The load or the work load is the demand on a system. The term load simply means the mix of demands placed on a system while we measure its performance and robustness characteristics. In

practice, most loads vary continually, so later we will address the challenge of determining the most appropriate load(s) for testing. The terms work load and benchmark are sometimes used as synonyms for load. A benchmark usually means a standard load, one used to compare the performance of systems, system versions, or hardware environments, but the benchmark is not necessarily the actual mix of demands at any one user installation. The term work load is a synonym for a load, and you see both of the terms in this book: they are interchangeable.

***Load Test*** Measuring a system's performance and resource utilization under load (usually heavy load). In contrast to a performance test, a load test is a measurement of performance under heavy load: the peak conditions. Because loads can have various sizes, more precise terms for this type of testing are peak-load testing or worst-case-load testing. A performance test may be run with a typical, representative load, but this measurement may not tell us much about the system's behavior under heavy load. For example, let's assume that the peak load on a system is only 15% more than the average load. The system may degrade gracefully – the system runs 15% slower at peak load. Often, though, the performance under load is non-linear: as the load increases by a moderate amount (in this case, 15%), the response time does not increase by a comparable percentage but instead becomes infinite because the system fails under the increased load. An imprecise shortcut to calculate the height of the peak load to use in testing: multiply the average load by a fixed factor – 3 to 5 times more for client/server; 10 to 25 times more for web sites with sudden spikes.

***Load Variation Test*** In this type of testing, the load varies during the performance measurement, to reflect the typical pattern of how the load ebbs and flows over time. This provides a more realistic picture of the system's performance characteristics than testing with a steady load for a duration.

Local area network. A network within a small geographic area like an office building or a college campus.

Localization test. See internationalization test. This term refers to making software specifically designed for a specific locality.

Locator. A software program which assigns physical addresses to the relocatable software produced by a linker. The result is an executable file.

***Log On / Off*** The user actions which initiate and terminate a session.

Logic analyzer. A hardware debugging tool which captures the hardware states and signals in real-time systems, for diagnostic and debugging purposes.

Logic test. See path analysis.

Loop. In a loop structure, the software presents a choice or asks a question.If the answer requires an action, it is performed and the original question is asked again until the answer is such that the action is no longer required. For example, a program written to compute a company's weekly payroll for each individual employee will begin by computing the wages of one employee and continue performing that action in a loop until there are no more employee wages to be computed, and only then will the program move on to its next action. Each pass through the loop is called an iteration. Loops constitute one of the most basic and powerful programming concepts.

Loop testing. Testing software or a process which contains feedback loops.

Loose-tight fit. The tight part means having strong core values to guide an organization. The loose part means that, once these strong core values have been established, the organization allows project teams considerable autonomy in carrying out their objectives.

Loss. An undesirable outcome.

Low-level design. The specification and design of the individual software components within the high-level design architecture, including the interface (input and output) and data storage of the component, and its expected behavior (algorithm). Also called detailed design.

Low-level integration. Connection of a string or hierarchy of atomic software components.

***Maintainability*** Can the system easily be modified or fixed? What is the feasibility and ease of making

modifications? How rapidly and how safely can enhancements and repairs be made? Maintainability is the degree to which software is robust, flexible and amenable to change.
Maintenance test. The procedure and issues in testing a maintenance change to an existing system.
Maintenance. The on-going activities to make modifications to software after it has been delivered to the end users.
Make-or-buy decision. The decision whether to build software internally, acquire a package, contract for its development, or adapt existing systems and re-usable components to form the software.
Manageability. Complex systems which operate in demanding environments need to be continually managed, either semi-automatically of by specialized system administrators. These systems include Web sites, operating systems, server clusters and networks. Manageability is the degree to which these systems can be managed prudently and efficiently.
*Manual Test Driver* In most test projects, at least some portion of the test cases are executed manually. The manual test driver is a keyboard, handset or some other test input device which can be used by a person. If an embedded system normally has no human input device, one will need to be built by the testers specifically for the manual testing.
Manufacturability. Can the product be manufactured? (Generally this not an issue with software.) The manufacturing process is relatively straightforward for most software, e.g., simply making and packaging copies of CD ROMs. For embedded products and integrated hardware/software products, though, the feasibility of manufacturing the integrated hardware/software product is an important factor in the overall system quality.
***Maximum Throughput Calculation*** Based on the measurements from the test tool and from the monitoring tools, you can calculate the maximum throughput that the system (used in the test) will support. This is essential information that will be used in the capacity planning.
McCabe's measure. The count of basis paths in a software module, also known as cyclomatic complexity.
***Mean vs. Median and Mode*** A reminder of our high school math is helpful here – remember how you went to sleep when the teacher was talking about the difference between average (mean) and median values? For example, since most transactions vary in length, data about their lengths can be expressed in terms of the mean, the median or the mode. We calculate the average by taking the combined lengths of all the transactions in a category and dividing by the number of transactions in that category. The median is the length of the transaction which lies midway between the longest and the shortest in the same group of transactions. (The mode, which we have not been given in this exercise, is the length of the most frequently occurring transaction.) If a set of data values follows a bell-shaped (Gaussian) curve and is symmetrical around the median value, this means that the median, mode and mean are all the same for that set of data. The range of the transaction lengths is usually not symmetrical around the median or the modal length, but is skewed. For example, if a few search response transactions are much longer than a median length of 2.4 kilobytes (KB) and are at the maximum upper limit of let's say 256 MB each, which is likely to be the case, then the average transaction length will be significantly more than the median value. In this case, the average length could have a value like 4KB or 8KB.
***Measurement Observed***, quantitative data about software or about a software engineering process. The gathering of quantitative data about software or about a software engineering process.
***Measurement of Delays (Latency)*** To be able to measure response times for particular events, we need to assume a straightforward cause-and-effect relationship: this stimulus triggers that outcome. In this situation, we can easily identify the stimulus for each outcome, and we can measure the delay from this particular stimulus to its particular outcome. If we cannot easily link the system outcomes to the stimuli, though, we need a more elaborate measurement (and model) of system behavior than the end-to-end response time. Consider a situation, for example, where the system logs and stores a stream of events in a file, but takes no action until the accumulated number of events reaches a certain threshold. We could reach this threshold within seconds or not for several weeks, depending on the work conditions.

In this situation, we may be interested in three elapsed-time numbers: (1) from the first event to the observable outcome, (2) from the very last event to the outcome, and (3) the average response time (from the median event to the outcome).

***Measurement of Losses*** In networks especially, losses are a way of life. In analog networks, signals can attenuate (weaken) and their wave shapes become corrupted. In a congested switch, blocking may cause a loss – all ports or connections into the switch are already busy, and the system simply drops an incoming message when the input hopper (buffer) is already full. In packet-switched digital networks, a data packet can be lost in transit. In the Internet, for example, a data packet is deliberately killed when the number of hops which the packet takes from node to node (i.e., from router to router) exceeds a threshold (usually 16 hops), in order to prevent them endlessly circulating within the Internet and royally gumming up the works. Elaborate facilities have been designed into the Internet to take care of these packet losses. Despite the fact that losses are considered routine, though, a high rate of loss is unattractive (usually anything more than a few percent). Every lost packet in the Internet generates at least two more messages, a request back to the point of origin to re-send the lost packet, and the re-sending of that packet. Thus the rate of loss tends to have a major impact on performance, and a NASA study found that a 3% loss of data packets in the Internet leads to a 50% degradation in throughput.

Measurement. (1) Observed, quantitative data about software or about a software engineering process. (2) The gathering of quantitative data about software or about a software engineering process.

Memory. Information storage facility for a computer.

***Memory Leak*** A bug caused by lack of sufficient memory (memory starvation), when an application continually requests portions of memory from the operating system but does not releases them. See allocation, de-allocation. A memory leak is the gradual loss of available computer memory when a program (an application/process or part of the operating system) repeatedly fails to return memory that it has obtained for temporary use. As a result, the available memory for that application or that part of the operating system becomes exhausted and the program can no longer function. For a program that is frequently opened or called or that runs continuously, even a very small memory leak can eventually cause the program or the system to terminate. A memory leak is the result of a program bug. Some operating systems provide memory leak detection so that a problem can be detected before an application or the operating system crashes. Some program development tools also provide automatic "housekeeping" for the developer. It is always the best programming practice to return memory and any temporary file to the operating system after the program no longer needs it.ii

Methodology. Organized process, guideline or template for how to accomplish something.

Metric. A measurement or a standard of measurement. Software metrics are the statistics describing the structure or content of a program. A metric should be a real objective measurement of something such as number of bugs per lines of code.

Microprocessor. A computer processor on a silicon chip, such as Intel's Pentium.

Middleware. Software which provides the data translation and conversion among other pieces of software, so that they link and communicate.

Milestone. A significant accomplishment, and the point in time when it is expected to occur or did occur.

MIS. (1) Management information system. (2) Management information systems group, sometimes called IS (information services) or IT (information technology).

Modular design. A system design which has modularity

Modularity. The characteristic of a system which has been designed as a set of discrete but connected components.

Module. A portion of a system. Also called a component or a software component.

***Monitor*** (1) To observe behavior. (2) A visual display terminal. (3) A debugging tool.

Monitoring. Observing the behavior of a system, usually without interfering with it.

Monkey. Software which performs simple tests.

# *Appendix A: Basic Definitions and Concepts*

Monkey test. A test that can be done without any knowledge about the internals of the system, let alone understanding the requirements or the business domain. The term “monkey” may be perceived as demeaning by the people who have to perform the tests, because it implies little skill or intelligence is needed. The difference between monkey testing and random testing, is that a monkey test assumes no information about the system, while a random test assumes that the randomness will mimic the actual pattern of user behavior. Monkey testing can be performed manually or automated, with an automated tool simulating a rapid succession of calls to the \component with random variations.

***MTBF*** Mean (average) time between failures, a measure of reliability.

MTTF. Mean time to failure. See MTBF (which is not the same thing).

Multimedia test. Testing of systems such as Microsoft's Flight Simulator, which may include video, graphics and sound, and all of which may be integrated with text and data.

Multi-processing. The use of multiple processors within a single coordinated system. A multi-processing system usually as common memory through which the processors share data and communicate with each other.

Multi-tasking. The concurrent execution of multiple software components. Each component (ands each copy of as component) which is running is a separate task or thread of execution. The operating system simulates parallel processing by dividing the processor's time among the running tasks.

Multi-threading. See multi-tasking.

Mutation analysis. Method of assessing the effectiveness of a test suite, by introducing small variations (mutations) into the software being tested, to find whether the test suite is sensitive to (i.e., can detect) the introduced changes.

Mutex. A flag used to by multiple concurrent tasks to signal mutual exclusion and coordinate their activities. Also known as a semaphore. See semaphore, mutual exclusion.

Mutual exclusion. A guarantee of exclusive access to a shared resource on a temporary basis. Shared resources in a system include a shared variable or a section of memory. The exclusion is signaled to other tasks by setting a semaphore or mutex.

***Nagling*** Automatic concatenation of several small buffer messages to increase the efficiency of a network application system by decreasing the number of packets that must be sent. Named for its creator, John Nagle, it relieves congestion in a TCP/IP network when an application generates data one byte at a time, causing the network to be overloaded with packets.

Navigability test. (1) Testing the navigability of a GUI application or a Web site, i.e., checking whether the visible linkages work. (2) In a graphic user interface (GUI) or windows environment, confirm the navigation is correct, i.e., that when an icon, menu choice or button is clicked on, the desired response in fact occurs.

Negative test. One where input data and the initial conditions are deliberately chosen which should <u>not</u> be acceptable to the system being tested, or a necessary condition is violated during execution. A negative test must result in an error message or an error recovery process, in order to be considered successful. The testing is aimed at showing software does not work. Also known as "test to fail" or destructive testing. See also Positive Testing.

Network. Interconnected set of computers or other communicating devices.

Network acceptance test. The purpose of these tests is to ensure that the data center and/or network administrators can safely proceed with the installation process, without inadvertently affecting other operations. Usually not performed by the users as part of their acceptance test, but performed in parallel by data center operations and network professionals.

Network readiness criteria. See network acceptance test

NIH. Not invented here.

Non-Functional Requirements. The types of non-functional requirements include the basics such as usability, security and performance, dependability, sustainability, fit with the environment, readiness to

implement, and operational concerns.

***Non-invasive Test*** One which allows the object being tested to remain pristine (usually preferable to invasive testing). However, non-invasive tests often cannot reveal as much as invasive ones, because probes cannot be placed into the object being tested. Also called non-intrusive testing.

Non-invasive test. One which allows the object being tested to remain pristine (usually preferable to invasive testing). However, non-invasive tests often cannot reveal as much as invasive ones, because probes cannot be placed into the object being tested. Also called non-intrusive testing.

***Number of Performance Measurement Cycles*** One of the major reasons for under-estimating the time and resources needed for performance testing, and thus seriously compromising the test results, is not allowing for sufficient iterations of the performance measurement. When a system is first delivered for performance measurement, it is very unlikely to be optimally tuned and in fact is probably de-tuned. This means that performance testing is not a one-time event. Based on the feedback from the performance test, the system will be tuned and its performance will then be re-measured. Performance tuning is a complicated activity, especially for complex systems. If the tuners change only one factor which affects the performance of the system at a time, before the system's performance is re-measured, we can establish a clear cause-and-effect relationship between the changed factor and the resulting change in the system performance. This approach to tuning is called OFAT (one factor at a time), and – while this is a common approach – it does not work very well. Unfortunately, even a relatively simple system such as a single cell phone usually has several factors which we can change to improve (or degrade) performance. In small networks, the number of distinct factors which affect performance can easily exceed one thousand, and in large networks the number becomes astronomical. This means, assuming a limitless schedule and budget, the number of cycles of performance tuning and re-testing could be very large indeed. In reality, deadline pressures and resource limitations temper the number of cycles. So how many cycles of performance measurement can we plan for? First, accept the fact that one cycle is unrealistic -- the performance is unlikely to be acceptable the first time around, and the system will need to be tuned and re-measured at least one more time. As a rough guide, most performance test teams plan for three to six iterations of performance testing in all. The number is usually higher in situations where performance is critical or where the system resources are severely constrained (e.g., we expect a low-power processor to handle an ambitious work load with hard real-time processing deadlines). In these performance-critical situations, the number of iterations can reach 20 or more.

***Object***. A software component which includes a set of related data and a set of processes on that data.

Object code. Machine-readable and executable instructions, the output of a compiler or assembler.

Object file. A file containing object code. The output of a compiler or assembler.

Object-oriented system test. Testing systems designed and coded using an objected-oriented design approach and development environment, such as Rational Rose, C++ or Smalltalk.

OFAT. One factor at a time. Refers to a performance tuning technique where only one adjustment factor is calibrated at a time.

OLAP. On-line application processing.

One-time programmable. A device, such a PROM, which be programmed only once and cannot be re-programmed.

On-line help test. Validating that the system's on-line help facility is accurate, complete and usable.

On-line test. The conversational, interactive nature of on-line testing means a different (interactive) test approach is needed for on-line testing than for batch testing.

OO. See object-oriented.

Opcode. Operational code: a machine-level instruction which can be directly executed by a processor.

*Open Problems Report* Lists and tracks the open problems (bugs).

Open source software (OSS). Community-developed software, where the developers publish and distribute the source code for free, and given permission for the source code to be enhanced by others.

# *Appendix A: Basic Definitions and Concepts*

***Operability Test*** Test that a system which works fine in a controlled, environmentally clean environment, also works in the real world (e.g., on a dirty factory floor or high in the ionosphere)
Operability test. Testing that a system which works fine in a controlled, environmentally clean environment, also works in the real world (e.g., on a dirty factory floor or high in the ionosphere)
Operating system & compiler test. The testing of support software, either by the vendor(s) or by sophisticated clients.
Operating system. The software which interacts with the hardware and allows applications to run on that hardware, such as Microsoft Windows and IBM's MVS. Operating systems manage the resources, and usually support capabilities like multi-tasking.
Operational. Pertaining to the live operation of a system.
Operational concerns. Requirements in this area address cost effectiveness, profitability, timeliness, saleability, quality of service, service level agreements, documentation, and manageability
***Operational Profile*** An operational profile (OP) is a list of the demands placed on a system, together with frequency of occurrence for each one. Depending on the situation, the demands could be events, transactions, messages, database accesses, etc. The operational profile of the test load should match the profile expected in live operation.
*Operational Profile Builder* A tool that builds a statistical model of the relative frequency of utilization of features, or the frequency of occurrence of events.
Operational profiling. Use of statistical techniques to develop a profile of the use of a system. Based on their expected frequency of actual use, determine which paths, components, conditions or data tables merit the effort of testing. If a path is expected to be taken extremely infrequently, for example, it may not be worth the test effort.
Oracle. The decision-making mechanism or tool, device or procedure that evaluates the test results and makes a pass/fail judgement. Usually does this by comparing the actual results of a test case with a reference file containing the expected results. May also contains logic to independently compute the approximate results expected from the test case.
Orthogonal array. Orthogonal arrays provide a subset of test cases to actually execute, from the set of all possible test cases. The method determines the minimum subset of test cases, so that all combinations of any two test factors are included in this subset of test cases. Telcordia's AETG and Lucent's OATS (Orthogonal Array Test System), are software tools whicht generate orthogonal arrays and test cases based on a description of the test factors.
OSS. Open source software.
Outage. Loss of service.
Outsider. A person who is not a member of the inner core team which develops or maintains a system. Outsiders can bring a fresh, independent perspective to the validation of the system.
Outsource. Contracting of software-related work, such as testing, to an external third party.
Overflow. (1) In a queue or buffer, an attempt to add more records than can fit in the maximum allowed capacity. (2) In a data filed, an attempt to write a value larger than the maximum number of characters allowed for that field.
Package test. See software package test
Panic. An unstable state in which a system thrashes.
Paper prototype. A mock-up on paper of a system (e.g., story boards which describe the user interaction with the system). See prototype.
Paradigm. A model.
***Parallel Process Test*** A test in a parallel processing environment, also called multi-threading or concurrent processing and not to be confused with traditional parallel processing. Usually requires multiple processors that are simultaneously processing and interacting, or at the very least a multi-tasking operating system. In this context, parallel processing means coordinated computing where multiple

interdependent processors are jointly working on the same problem. Parallel process testing is not the same as parallel testing. Parallel process testing is not the same as parallel testing.
Parallel processing. The ability to use two or more processors on interdependent parts of the same computation.

***Parallel Test*** Parallel testing has two main forms: A before-and-after comparison, to assess the impact of a change to an existing system. A side-by-side comparison, to help ensure that a replacement system adequately replicates the behavior of the one being replaced. A full-fledged parallel test, which is unusual because of the expense involved, includes duplicate manual data entry into the two systems. A full parallel test can be very expensive, because to some people it deliberately includes dual data entry into the parallel systems, and dual processing for a period of days or weeks. This dual effort requires up to twice the operational and support staff during the time of parallel operation. Like volume testing, parallel testing usually looks for features which fail under load, even if the system keeps running, through the before-and-after comparison of outcomes. Parallel testing is also called comparability testing.

***Pareto (80-20) Analysis*** A test focusing strategy. Analyze the defect patterns and identify their causes and sources or likely sources, and rank them by frequency of occurrence and criticality based on the 80-20 rule. Add test cases to check if the most likely or risky causes result in defects. Focuses the testing on those locations or parts of the system where the prior incidence and cost of defects have been highest. ("Hunt for the bugs where the critters hang out.")

Parse. Scanning or reading an input string; for example, a compiler parses a source language instruction.
Path analysis. (1) A technique to develop test cases, based on the decision logic of a process or a piece of software. The intent is to test each major decision option (i.e., each distinct path on the flow chart). With variations on the theme, this technique is also called "branch coverage", "control-flow" and "logic coverage" testing. (2) A method of identifying tests based on the flows or paths that can be taken through a system. It is also called branch testing, because the branches through the code or functional logic are identified and tested.
Path Testing. Testing which uses path analysis
Pattern. A model or template that can be used in making things.
PDA Personal digital assistant.

***PDC*** (***P****rimary* ***D****omain* ***C****ontroller)* A server that maintains a read-write directory of user accounts and security information. The PDC authenticates usernames and passwords when members log into the network. Members only have to log into one domain to access all resources in the network.
PDL. Problem definition language. These usually combine natural language with programming language-like constructions.

***Peak*** The size or volume of the peak demand. Usually the relevant events are counted for an interval of time (e.g., per hour or per millisecond), so the peak depends on the granularity or length of this interval – more happens in an hour than a millisecond. In addition, the peak interval is the highest one of several, so the peak also depends on the overall time duration (i.e., how many time intervals there are in the duration). The peak hour in a typical week usually has a higher volume than the peak hour in a typical day. By the same token, the peak hour in a typical month usually is higher than the peak hour in a typical week, and so on. The longer the time period, the higher is the peak demand during that time period.
Peer review. In a peer review, a software work product is presented to the producer's colleagues to identify defects.

***Penetration Test*** Act an individual who is intending to infiltrate the security of an application. This includes testing not only the application itself and it's security features, but also those of the network, hardware, and physical location where the product is running.

***Percentile*** The nth percentile is a value so that n% of the data is smaller and (100-n)% of the data is larger. Percentiles can be computed for ordinal, interval, or ratio data.

# *Appendix A: Basic Definitions and Concepts*

***Performance*** A system's performance is its speed or responsiveness, its ability to handle loads and its efficient use of resources. Depending on the situation, the term performance can mean response time, throughput, availability, error rate, resource utilization, or another system characteristic (or group of them), which we are interested in measuring. "All promise outruns performance." Ralph Waldo Emerson. Performance testing involves the testing of the product under stress, maximizing the load of the application or volume testing and timing the ability of product to handle capacity or load over a specific period of time. Any testing designed to understand the performance dynamics of the product. Products sometimes have specific performance or efficiency objectives, stating such properties as response times and throughput rates under certain work load and configuration conditions. If these requirements are defined, then it is important to exercise these performance requirements as stated and beyond.

***Performance Engineering*** Designing systems to meet performance requirements.

*Performance / Load Test Driver* These tools are used to drive performance and stress tests. The tools "pump" a large volume of test cases through a system.

***Performance Management*** Performance management is monitoring and troubleshooting on-going performance in live operation.

*Performance Measurement Tool* Captures response times, throughput, system availability, and other measures of SUT performance.

***Performance Profiling*** Measures the performance in the system under test and determines the contribution of individual components to the overall performance.

***Performance Test*** The purpose of performance testing is to measure a system's performance under load: the testers are normally interested in both "how much?" and "how fast?" As Humpty Dumpty said, a word can mean whatever one chooses it to mean, so it is worth our time to examine what we mean by the words "measure", "performance" and "load". Performance testing is a measurement of performance characteristics, although sometimes the use of the word "testing" confuses people. Some performance professionals feel strongly that it is important to not use the term "performance testing", but to call it performance measurement instead. They are concerned that this measurement will get confused with feature testing and debugging, which it is not. They point out that measurement is only testing if the collected measurements are checked against pre-established goals for performance, and that measurement is often done without preconceptions of required performance. These people have a good point: clarity of terminology is important. But since most people use the term "performance testing" we will go with the majority and use it too. Performance testing simulates the typical user experience under normal working conditions. The load is a typical, representative mix of demands on the system. (And, of course, there can be several different representative loads -- the work load at 2 p.m., at 2 a.m., etc.) Another name sometimes used for a performance test is a capacity test, though there is a minor difference in these terms as we will see later. First, the performance testers need to define what the term performance means in a specific test situation -- that is, what the objectives are and what we need to measure in the test. The answer to this question is that we measure performance usually as a weighted mix of three characteristics of a system: throughput, response time and availability. In real-time systems, for example, the users need a guarantee that a task will always be completed within a fixed time limit. Performing a task correctly but a millisecond too late could literally be fatal.

***Performance Test Plan*** A performance test plan tells the who, what, when, why, and how of the testing effort. It lists specific tasks assigned to specific people to be completed by specific dates with specific expected results.

***Performance Test Strategy*** A high level approach to the testing effort that demonstrates how the goals of the testing effort will be achieved by answering the question "What do we do when…?" without the constricting details of a detailed test plan. Test Strategies are generally complimented with weekly or bi-weekly "mini-plans" that detail the immediate next steps of the effort.

# *Appendix A: Basic Definitions and Concepts*

Peripheral. A piece of hardware attached to the processor in a system, typically memory or an I/O device.
***Pilot Test*** A field test at a limited number of sites, prior to widespread deployment or distribution of the product. A pilot differs from a beta test, in that a beta is done by a vendor using prospective customers, whereas a pilot is usually conducted in-house. Placing a new system in limited actual live use, at a small number of field sites, prior to a full deployment of the product.
Pilot test. (1) A field test at a limited number of sites, prior to widespread deployment or distribution of the product. A pilot differs from a beta test, in that a beta is done by a vendor using prospective customers, whereas a pilot is usually conducted in-house. (2) Placing a new system in limited actual live use, at a small number of field sites, prior to a full deployment of the product. See: beta test.
Plan. A scheme or strategy worked out beforehand for the accomplishment of an objective.
*Platform Management Tool* These tools perform configuration management on the test platform, load new versions of the SUT, and provide diagnostics on the test platform.
Polling. A method of determining when software tasks or hardware devices need service from a processor without using interrupts, by repeatedly reading a status indicator until the task is ready. Used in situations such as the space shuttle where the uncertainty of interrupt timing delays cannot be tolerated.
Portability. The capability to migrate or transport software from one target environment to another.
Portal. Gateway, point of access.
Positive test case. One where input data and the initial conditions are deliberately chosen, which should be acceptable to the system being tested. A positive test exercises a “normal”, mainstream use of the system.
Positive Testing. Testing aimed at showing software works. Also known as "test to pass" or mainstream test. See also Negative Testing.
Pre-emptive. The ability to temporarily suspension a running task when a higher-priority task needs service. Generally requires a multi-tasking operating system or the ability to process interrupts.
Primary Domain Controller See PDC
Prior defect history. A test focusing strategy. Devise and add a test case for every defect found in prior versions of the system or in other comparable systems, or at least for every defect above a certain level of severity. This method is sometimes called fault-based testing, because it focuses on finding defects based on the patterns of previous defects.
Priority inversion. A situation in which a high-priority task is delayed while waiting for access to a shared resource which is tied up by a lower-priority task or is not even being used at the time. This undesirable behavior happens occasionally in operating systems or other control software.
Priority. The relative importance of a pending task, such as a bug fix, compared to other similar tasks.
Problem report. Written description of a failure, used to initiate the follow-up debugging and fixing activities.
Problem. An issue which needs to be resolved.
Procedural design. Development of the procedural logic or algorithm for a software component or system.
Procedure. A documented step-y-step guideline on how to perform a task it in a reliable and consistent way to obtain a specified outcome.
Process. (1) A systematic method for performing a work activity. See procedure. (2) A series of actions, events or phases which takes place over time and has an identifiable purpose or result. (3) In a computer system, a process has its own private memory space. By contrast, the tasks or threads in a system share a common memory space.
Process description. A narrative explanation of how a process works, such as the logic of a software component.
Process flow. A visual or narrative description of how a process works in a sequence of steps, and showing the relationships among those steps.
Process tailoring. The activity of elaborating or adapting a process to a particular context.

# *Appendix A: Basic Definitions and Concepts*

Processor family. A group of related processors, such as the successive generations of Intel x86 chips.
Processor. The hardware which executes software instructions.
Processor-independent. Software which is independent of the processors on which it runs. This independence is accomplished by a compiler, which translates processor-independent source code into processor-specific object code, or by a virtual machine such as the Java virtual machine (JVM).
Processor-specific. A piece of software that localized to a specific processor and which not run on other processors without major modification. Contrast with processor-independent.
***Production Test*** A test which addresses these questions: Does the system operate correctly in on-going production? Is the on-going monitoring and evaluation of the production results happening adequately?
Production test. A test which addresses these questions: Does th system operate correctly in on-going production? Is the on-going monitoring and evaluation of the production results happening adequately?
Productivity. Work accomplished per unit of time.
Profiler. A software tool which builds a profile of how software is used, e.g., which paths through a software component are taken and how frequently, which other components are called and the total amount of time spent in each one, etc.
Profitability. Cost effectiveness is important for a vendor to be competitive, but not if the firm is losing money on each sale.
Project. An organized effort to accomplish a goal. Synonym: venture.
Project control. The control of budget, schedule, risk, quality, scope and change on a project.
Project plan. A description of the technical and management approach to be followed on a project. The plan typically describes the work to be done, the resources required, the methods to be used, the procedures to be followed, the schedules to be met, and the way that the project will be organized.
Project risk. A significant chance of a loss associated with a particular project.
Project tracking. Monitoring the activity and determining the status of a project.
PROM. Programmable read-only memory. A type of ROM which can be written (programmed) with a device programmer. Also called write-once or one-time programmable (OPI) devices.
Prototype. A working model or a mock-up of a system, used to refine the requirements and design, and to perform early usability testing. Usually emphasizes the look-and-feel of the system but does not contain all its internal logic or controls. *Prototype / Simulator* tools also can be “stand-ins” for the SUT. They are used to check out the test environment, and to begin testing, before the SUT itself is available for testing.
***Prototype Test*** Building prototypes for proof-of-concept testing, and also testing prototype systems if and when they are ramped-up to become fully operational production systems.
Prototyping. The process of developing and evolving prototypes, usually in close conjunction with the users or their surrogates.
Provably correct software. Software which can be proven to be correct through a mathematical theorem.
Proxy server A server that sits between a client application, such as a Web browser, and a real server. It intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server. Proxy servers can dramatically improve performance for groups of users. This is because it saves the results of all requests for a certain amount of time. Proxy servers can also be used to filter requests. For example, a company might use a proxy server to prevent its employees from accessing a specific set of Web sites.
Quality. User satisfaction: a system has quality when it functions as the client or user can reasonably expect.
QA. Quality assurance.
QC. Quality control.
Quality assurance (QA). (1) Independently verifying the quality of a system. (2) Quality activities which are work process oriented. (3) All those planned or systematic actions necessary to provide adequate

confidence that a product or service is of the type and quality needed and expected by the customer.
Quality audit. A systematic and independent examination to determine whether quality activities and related results comply with planned arrangements and whether these arrangements are implemented effectively and are suitable to achieve objectives.
Quality circle. A group of individuals with related interests that meet at regular intervals to consider problems or other matters related to the quality of outputs of a process and to the correction of problems or to the improvement of quality. Quality circles are periodic, organized brain-storming sessions where the members of a work team mutually examine their practices and develop improvements.
Quality control (QC). (1) The work done by insiders to build quality in and to test a system. (2) Quality activities which are work product oriented. (3) The operational techniques and the activities used to fulfill and verify requirements of quality.
Quality gateway. A checkpoint where the quality of a deliverable is assessed, usually by using a standard checklist or procedure.
Quality management. The overall management of the quality function the quality policy.
***Quality Metric*** A measure of quality.
Quality of maintenance. (1) Likelihood that enhancements and fixes work correctly. (2) Introduction of inadvertent new defects. (3) Timeliness of maintenance.
Quality of service. QoS is often negotiated between providers of services and their customers, such as a telecommunications carrier and a nationwide bank. Financial penalties are incurred if the QoS is not met. QoS is a single combined measure, or a small number of measures, which includes elements such as response time, ability to handle volume, availability, error rates and recovery time, for different types of data and different types of user. QoS measures tend to be industry-wide benchmarks, a way of measuring and comparing quality as a number or set of numbers. SLAs by contrasted are usually individually negotiated between a particular client and supplier.
Quality policy. The overall intentions and direction of an organization as regards quality as formally expressed by top management.
Quality system. The organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.
***Race Condition*** A situation where related events do not occur in the required sequence needed to produce the desired outcome, i.e., the outcome is affected by the exact order in which the instructions are executed. Race conditions cause concurrency problems, e.g., multiple accesses to a shared resource, at least one of which is a write, with no mechanism used by either to moderate simultaneous access.
RAD. Rapid application development.
***RAID*** Redundant array of inexpensive disks, a method of organizing disk drives to improve database performance and reliability.
RAM. Random-access memory, where individual memory locations can be accessed, read or written as needed.
Ramp test. Continuously raising an input signal until the system breaks down.
***Ramp-Up Test*** The term “ramp-up” is used in two ways. In the first use, it measures the time need to initiate a process. For example, let’s say that a system contains an automated file back-up process which is triggered under certain circumstances, such as when the file size reaches 90% of the available storage capacity. Let’s also assume that the system requirements state that the back-up process must launch within ten seconds of the file size reaching the 90% capacity. We’d use a ramp-up test to confirm that this process launch happened as expected and that the system meets the timing requirement. The second way the term ramp-up test is used is similar to a breakpoint test, in that we observe the trends in response time, resource utilization, etc., as the load increases (i.e., ramps up), although we do not necessarily increase the load to the point where the system fails.
***Ramp-Up / Ramp-Down Time*** Ramp-up is simply the time between when the first user accesses your

application and when the target load of your test is reached. Since we are interested in the performance of the system only during the time between the start and end points, no performance data from outside of that time will be collected and since a user obviously can't start in the middle of their activity on the site, all of the green users need to be included. The ramp-up time becomes the time between the beginning of the first user activity and the start of the model, and the ramp-down time becomes the time between the end of the model and when the last user completes their activity.

Random generation of input data. Using a test data generation tool, generate a sizeable volume of input data. This data is simulated, not live data. The tool can generate data to fit desired profiles, for example, to reflect the demographic mix of actual live data. If a tool is not available or only a small volume of test data is needed, have people bang away on the keyboard in quasi-random patterns.

***Randomization*** The process of randomly varying test data, e.g., selecting the names of people for a series of queries.

Range test. For each output, identify each domain over which the system behavior should be the same. Test one representative value within this range, and assume that its pass/fail result correctly predicts the results for all other values in the range.

Rapid application development (RAD). System development style which is intended to speed delivery.

Rate monotonic analysis. Method for calculating the upper limit on latency based on a system's design (the execution path with the longest duration).

Real-time operating system. An operating system designed for use in real-time systems. Usually RTOS are stripped down to run very fast and add minimal overhead to the work being processed.

Real-time system test. Testing that a real-time system works, i.e., a system where timing issues are critical and where a guaranteed response time is needed (for example, a system that automatically flies an airplane).

Real-time. An action which must be completed within a hard deadline, a deadline that must be guaranteed to be met in live operation.

Recoverability. If the system does crash, are the re-start and recovery mechanisms efficient and reliable? If the system does produce errors, are there mechanisms to help detect and recover from these errors, and minimize their impact? Recovery testing confirms that the system recovers from expected or unexpected events without loss of data or functionality. Events can include shortage of disk space, unexpected loss of communication, or power out conditions.

***Recovery Test*** This type of testing takes into consideration what will happen if there is a crash or other problem with the product. Can the system or product recover. Are there back-up procedures in place and do they work. Some of the problems can be a hardware failure, memory parity errors, I/O device errors or data errors. For example, unplug the Ethernet cable in the middle of an operation to determine how well the application will respond, and what state the data will be in as a result.

Recursive. The characteristic of a software component which allows to call itself without interference. The different calling and called versions of the component usually communicate through a stack.

Re-engineering. The re-furbishing of a legacy system with poor maintainability into maintainable software.

Re-entrant. The characteristic of a software component which allows it to be executed multiple times simultaneously without interference.

Refactoring. Selective re-writing of code to improve its flexibility and maintainability.

***Referential Integrity (RI)*** Data consistency; third normal form. RI is a feature of many DBMS which check that different storage locations do not contain conflicting data. Some DBMS do not provide the same level of RI across a distributed database as when the entire database resides on a single server. Other DBMS provide this feature, but because of performance consequences, it is not enabled by the DBA.

Register. A specialized, dedicated memory location which is reserved for use by a processor or an I/O device.

# *Appendix A: Basic Definitions and Concepts*

***Regression Test*** A re-test of a system, subsystem or component, after a modification has been made, to ensure that faults have not been introduced or uncovered as a result of the changes.
Release candidate. A pre-release version, which contains the desired functionality of the final version, but which needs to be tested for bugs (which ideally should be removed before the final version is released).
***Release Test*** A comprehensive test of the new version or upgrade of a system, prior to the release of that new version.
Release. A version of a system that has been or will be migrated (released) to live operation.
***Reliability***. The likelihood that a system will execute for a period of time without error.
***Reliability Percentage*** Given that we make statements about a system that has a planned availability of 24 hours a day 365 days a year, the following applies: 99% reliability means the system s down approximately 3 days and 5 hours per day, while 99.999% ("five nines") means approximately 5 minutes.
***Reliability Test*** Assessing whether a system meets (or will meet) its reliability goals, as measured by availability, mean time between failure (MTBF) and mean time to repair (MTTR).
Relocatable. Software containing instructions which have relative, not fixed addresses, so that the execution of the software is not dependent on the particular memory location into which it is loaded.
***Remote Procedure Call (RPC)*** A communication model where requests are made by function calls to distributed procedures elsewhere. The location of the procedures is transparent to the calling application. RPC is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. (A procedure call is also sometimes known as a function call or a subroutine call.) RPC uses the client/server model. The requesting program is a client and the service-providing program is the server. Like a regular or local procedure call, an RPC is a synchronous operation requiring the requesting program to be suspended until the results of the remote procedure are returned. However, the use of lightweight processes or threads that share the same address space allows multiple RPCs to be performed concurrently. When program statements that use RPC are compiled into an executable program, a stub is included in the compiled code that acts as the representative of the remote procedure code. When the program is run and the procedure call is issued, the stub receives the request and forwards it to a client runtime program in the local computer. The client runtime program has the knowledge of how to address the remote computer and server application and sends the message across the network that requests the remote procedure. Similarly, the server includes a runtime program and stub that interface with the remote procedure itself. Results are returned the same way.
***Rendezvous Test*** This is a type of spike testing where many events "rendezvous" (i.e., happen simultaneously). For example, we can use a software tool to simulate a hundred users hitting the enter keys on a hundred keyboards at exactly the same time. Rendezvous tests are often unrealistic, because a hundred users are unlikely to hit their enter keys simultaneously. If we allow a small spread of events across time, instead of requiring an instantaneous happening, then the rendezvous test becomes much more realistic -- for example, what happens if a hundred users all hit their enter keys within a duration of two seconds?
Repository. See database.
Requirement. The description of an expected behavior of a system. Synonyms: Specification, functional specification, system requirement, user requirement.
Requirements analysis. The process of understanding the users' needs for a system, developing models of the system's behavior, and defining the requirements.
Requirements engineering. Requirements analysis, management and validation. Requirements in this area include functionality, testability, debuggability, installability, scalability, upgradeability and manufacturability
*Requirements Manager* These tools enable business analysts, system engineers, marketers and users to

compile, catalog and maintain lists of requirements.
Requirements management. The process of managing a repository of requirements, including versions of the requirements for different versions of the system, change approval and provision of audit trails of the changes to the requirements. Usually done with software tools which are called requirements managers.
Requirements validation. The process of reviewing, correcting if necessary and approving the system requirements.
Re-set address. The memory address from which the first instruction is read after a processor is powered on or re-set.
Re-set. The process of returning a system to a known pristine state ready for further use, or of re-starting or re-booting the system.

Resource. (1) Anything which is needed to accomplish a task or a project, including people, software tools, hardware and networks, documentation, etc. (2) Anything which is needed to run a computer system, such as a hard drive.
***Resource Utilization*** Monitoring the levels of utilization of system resources provides insights into how the system works (which may not be the same as how its designers think it works). It helps to identify bottlenecks, assess spare capacity and the potential for scalability, and how to improve the efficiency of the system. The resources and events which are monitored can include processor activity, use of cache memory and hard disk accesses, I/O traffic, page swaps, lengths of queues, overflows, number of ports which are busy, network bandwidth utilization, and number of concurrent software threads or processes which are running. Monitoring the resource utilization means we need access to the system logs which are recorded by the operating system, network management system, and database management system. Plus – and this is an important plus – we need to know how to read these logs. Often the numbers of entries in these logs are so voluminous that it's a good idea to use software tools to edit, extract and summarize the meaningful information. Although they are voluminous, these logs generally do not provide everything we need. In addition, we may need home-built or third-party plug-in tools to place probes into the system under test and gather the data, hopefully without materially changing the system's performance and robustness characteristics.
***Response Time Test*** This testing measures how long the system takes to complete a task or group of tasks. It usually represents the user viewpoint, i.e., we measure the likely delay as perceived by an external user. We also can measure the efficiency of an internal software activity or hardware component which is not directly accessible by the user. Response time is the total end-to-end elapsed time, which includes wait time in a queue prior to processing, and service time (the actual time to process the request for service). Wait time and processing time both can vary, and may be affected by different factors.
***Response Time vs. Throughput*** We generally need to measure both response time and throughput, because one does not necessarily predict the other. It is possible for a system to have fast responses with low throughput, or vice versa, slow responses with high throughput. There is not any iron law governing the trade-off of response time and throughput – it is not a zero-sum game. We usually are interested in achieving fast responses with high throughput. Part of the confusion between response time and throughput is caused by mis-using the word "fast". A Boeing 747 is not three times faster than a Boeing 737. They both cruise at a speed of around 500 miles per hour. The difference is in capacity -- the 747 carries about 500 passengers, whereas the 737 only carries about 150. Throughput depends on capacity (e.g., bandwidth) as well as transmission speed. The Boeing 747 is three times larger than the Boeing 737, but not any faster. If we want to go from New York to London, the Boeing 747 is not going to get us there three times faster than the 737. The end-to-end "response time" for any one traveler on the New York -- London flight is the same. On the other hand, if we had to transport 1,500 people to London and we could use only one airplane, the 747 could do it in three trips while the 737 would take ten, because the capacity and thus the effective throughput of the 747 is higher. Speed, capacity and

throughput may not be simple trade-offs. Suppose, for example, that an end user wants to transfer very large files more quickly. While the user wants to speed things up, the transfer is not urgent. A high-capacity, relatively slow communication link might appear to be the best thing for the job, if the file bulk is huge but the transfer is not highly time-sensitive. What the end user does not see is that, in order to make the file transfer, his or her computer has to send dozens of little control messages back and forth. In this situation, both the response times for the many small messages and the throughput for large files are important in fulfilling the user's need. We usually measure response time and throughput at different locations within a system, and may require different measurement tools for response time vs. throughput. With a web site, for example, one visitor to the site will know his or her own response time but may have no idea of what the total number of visitors and the total throughput are. By contrast, a measurement of throughput at the central Web server will give no direct indication of the response time at each visitor's remote workstation.

Re-start. Initializing and starting a system after a planned outage or unplanned crash, hang or panic.

***Results of Performance Test*** Most often we see results that meet or exceed the required levels of scalability. Knowing the actual break point eliminates the tendency to persistently over-engineer Take appropriate action to avoid the break point More efficient and better growth decisions are made by understanding the current system limitations.

Re-usability. The ability of a software component, test case or other item be adapted for new uses, so that new components do not need to be built.

Re-usable component. A component which is designed and built to be re-usable. The component may be a software module, test case, section of documentation or other item.

Reverse engineering: (1) Recovering the design or the requirements from the software code. (2) Building a comparable system based on its external behavior and without access to the internal design, to avoid legal infringement.

Review. An inspection or examination with the intention of evaluation.

RISC. Reduced instruction set computer. A processor design which gains speed by having a limited set of instructions. Contrast with CISC.

Risk anticipation. The act of identifying risks sufficiently early that reasonable corrective actions are possible.

Risk assessment. The act of determining the likely outcomes associated with a risk or a set of risks, and evaluating their likely probabilities and their likely costs.

Risk containment. The act of minimizing and controlling a risk factor after it has been identified and assessed.

Risk management. The process of eliminating or minimizing the potential negative consequences of risk, i.e., to improve the likelihood that all the remaining consequences of events are acceptable.

Risk monitoring. The continuous process of watching and re-assessing risk. Since conditions change, on-going monitoring is an important part of risk management.

Risk quantification. A risk assessment technique which depends on quantifying the probabilities and costs of outcomes.

***Risk*** A significant chance of a loss. Synonyms: threat, vulnerability, weakness, hazard, exposure.

Risk-based prioritization. A test focusing strategy. Identify the major risk factors that could occur during the system operation, and the conditions that will cause them. Test each of the related conditions and verify that the risks are averted or managed successfully.

***Risk-Based Test*** An approach where tests are identified and prioritized by risk, vulnerability or exposure. This method uses a risk assessment to identify and prioritize the likely risks which the system faces in live operation. We use this risk assessment to allocate test resources to the various aspects of the system, i.e., to focus the test effort to the areas which need the depth and intensity.

Risk-based testing. An approach where tests are identified and prioritized by risk or exposure.

# *Appendix A: Basic Definitions and Concepts*

Risky event. A significant event which is associated with a particular risk. A risky event either (a) provides an early warning that the undesired outcome is going to materialize, or (b) causes the undesired outcome to materialize, or (c) is the realization and confirmation of that risk, i.e., the undesired outcome actually occurs. Synonym: risky incident.

***Robustness*** Ability of a system to "take a licking and keep on ticking" because of overload, equipment malfunction, inadvertent operator error such as invalid input, malicious act, natural disaster, etc. If the system does fail, robustness includes the recoverability. May be measured or predicted in reliability terms of MBTF and MTTR. Also called dependability, survivability.

***Robustness Test*** Checking whether a system adequately prevents, detects and recovers from operational problems such as downed network connections, data bases which become unavailable, equipment failures and operator errors. Also called rainy day testing as opposed to sunny day testing.

Robustness test. Checking whether a system adequately prevents, detects and recovers from operational problems such as downed network connections, data bases which become unavailable, equipment failures and operator errors.

Robustness. Are automatic error detection and recovery mechanisms built in, to try to keep the system operating no-matter-what? Is the system structured in order to minimize the possibility of introducing new errors through bad fixes?

ROM. Read-only memory. Memory which can be be read but not written.

Roman arches method. Testing methods are not always a matter of mechanical techniques: human behavior also has a role. The ancient Romans supposedly built and tested arches using the following method. First, the arches were built using temporary timber scaffolding to hold the heavy stones in place, until the keystone at the top center could be inserted. The scaffolding was then removed to leave the finished arch. The arch was tested by requiring the builder to stand directly under the keystone as the scaffolding was removed. In the case of an unsound design or construction, the arch could fall and maim or kill the builder. If the builder moved from under the arch before all scaffolding had been removed, he was automatically sent to gladiator training for the follow-up assignment of feeding the lions in the coliseum.

***Round Trip Time (RTT)*** RTT, expressed in milliseconds, is the elapsed time for a request to go from node `A' to node `B', and for the reply from `B' to return to `A.' The RTT is the total time for the trip. The forward and reverse path times do not need to be the same. RTT depends on the network infrastructure in place, the distance between nodes, network conditions, and packet size. Packet size, congestion, and payload compressibility have a significant impact on RTT for slower links. Other factors can affect RTT, including forward error correction and data compression, which introduce buffers and queues that increase RTT.

RTOS. See real-time operating system.

***Safety test*** Testing intended to show that a system's safety controls work and that the system cannot do anything dangerous.

Safety-critical system test. Testing that life-critical systems are trustworthy. See safety.

Saleability. The marketer says cynically: "We don't care how technically ingenious it is. Can we sell this product?" This is an important issue for market-driven software vendors (and all vendors ultimately are market-driven). Saleability is usually defined as a mix of some of the prior dimensions, such as: provision of features which users like and want to use, ability to attract buyers, ability to fill a niche which is not already dominated by entrenched competitors, and time to market.

***Sample Size*** Most factors which we want to measure, such as response time, vary based on many factors – both known and unknown. This means we cannot measure something once but must sample it several times in order to compute a meaningful average. Performance testing is not complete until there is a sufficient quantity of trustworthy information to answer the clients' questions about the system performance. How big does the sample need to be? According to statistical theory, we need a minimum

sample of 30 measurements for each type of demand on the system, in order to derive reliable conclusions about average response time, etc. In areas where we expect a high variability in performance measurements, such as the Internet, the minimum sample size for the major transactions could be 100 or more..

***Sampling*** A way to minimize the testing effort, by allowing us to draw conclusions about large populations from the results of testing only a relatively small portion of the total population.

Sampling. A way to minimize the testing effort, by allowing us to draw conclusions about large populations from the results of testing only a relatively small portion of the total population.

Sanity check. A last-minute, informal double check, after testing, debugging and fixing, to make sure everything looks OK prior to system delivery.

Sanity test. Brief test of major functional elements of a piece of software to determine if its basically operational. See also smoke test.

***Scalability Test*** Checking that a system can work effectively with both small and large numbers of users, small or large databases, typical or heavy load, etc. This type of testing investigates a system's ability to grow. Growth can occur in several ways, which we may need to separately test: increase in the total load; increase in the number of concurrent users; increase in the size of a database; increase in the number of devices connected to a network, and so on. We can test systems for their ability to scale down as well as up. For example, we may be interested in this question: can the software run adequately on a cheaper, slower processor or with less memory?

Scalability test. Checking that a system can work effectively with both small and large numbers of users, small or large databases, typical or heavy load, etc. Scalability testing focuses on ensuring the application under test gracefully handles increases in work load.

***Scalability*** Can the system work effectively for one user, ten users or a thousand users?

SCCB. Software configuration control board, the group who makes decisions about changes and new releases.

***Scenario-Based Work Load*** A scenario-based work load is the most complex work load type. The objective is to replicate real-world conditions by varying the number of virtual users run during a load test, depending on the time of the day. This work load type is used for stability tests that run for extended periods of time - 24 hours, a few days, or even a week.

Schedule. The sequence of tasks or activities, the time line and milestones for the work on a project.

Scheduler. The part of an operating system which determines decides which task to run next, based on the readiness of the tasks and their relative priorities.

SCM. Software configuration management.

Scope. A statement of specifically what is included and what is not included in a project, such as a testing project, or in a system.

SDD. (1) Software design and development. (2) Software development documentation.

Security controls test. Testing the internal application controls, system access controls, and back-up, integrity and reliability features. Security testing confirms that the system can restrict access to authorized personnel and that the authorized personnel can access the functions available to their security level.

***Security Test*** Security testing is the process of attempting to devise test cases that subvert system security checks. Security testing is a very important aspect of testing, especially when it involves financial transactions or personal data (such as SSN).

Security. Assurance that a system, network or database cannot be accessed or used without the appropriate authorization.

Semaphore. A data structure that is used for coordination of concurrently running tasks. See mutex.

Service level agreement. Agreement between the provider of services and the user, typically addressing performance, reliability, allowable error rates, cost effectiveness and support. (SLAs) are operational support goals which are documented as agreements among internal IS departments and their clients, or

between service provider vendors and their clients. These agreements define service objectives and specify the agreed-on levels of service to be provided, based on the balance between (a) the resources and activities under the control of the provider and (b) the needs of the client.

***Session*** The series of individual events or activities (usually the externally visible interactions between one user and the system), which occur from the time when the user logs on to the system until the same user logs off.

***Session Duration*** Same as session length.

***Session Length*** The total amount of time that a single user is using web site during a single visit to that site (expressed in minutes fractions of an hour). Hourly users divided by average session duration results in a heavily averaged estimated of concurrent usage.

Severity level. The importance of a failure or defect. The severity level is often used to set the priority for follow-up debugging and fixing**.**

Side effect. A new error which is introduced, or an old error which is exposed, by a change.

***Simulation*** Simulation strives to mimic the production use of the systems as closely as possible by creating a performance model that contains all the ingredients that make up "normal" usage. By running this model under different load scenarios you can measure and assess the results. Simulation is the first technique available for applications that are newly developed. This technique is typically the first one deployed in a Performance evaluation exercise. It will generate the first set of results on the performance of a system, results that can be used with the other, here mentioned, techniques. Simulation is the process of using a computer program to model the actions of a user or system. In a load test the real user of a computer program is simulated, or modeled, by creating a virtual user, which then emulates the interactions of a real user with the system or program during the test.

***Simulation Test*** Testing simulation models of real-world situations, such as a forecast of the financial markets or the weather. The term simulation test is also used to mean testing a simulation of a system before the system itself is ready for testing, in order to check out the testing facilities.

Simulator. A system which mimics another system, for prediction, development testing or debugging.

***Simultaneous Users*** The start of execution/invocation of a particular sub-system/task/feature of the SUT by a pre-specified number of users (the work load) within the same brief time window (1-3 seconds) is termed as simultaneous usage of the system. This is usually accomplished by employing timing and synchronization features in work load generation tools - e.g. rendezvous points in LoadRunner enable one to synchronize users in a scenario for simultaneous usage. Caveat Emptor: Simultaneous usage may not always be reflective of actual usage and is to be used sparingly.

Six sigma quality. A very high standard for manufacturing quality, based on an allowable level of failure of 3 parts per million.

SLOC. Source lines of code.

Smart monkey. Automated test tool which has some knowledge of the functioning of the AUT.

Smoke test. A “quick and dirty” initial test, to make sure the system can operate and process at least a minimal workload. The term “smoke test” comes from the saying by hardware developers: “Power it up and see if any smoke comes out.” In complex systems where the build or integration process is not straightforward, a smoke test is a quick post-build test to ensure that all the components have successfully been integrated together. Its purpose is to provide assurance of the build process before proceeding to more comprehensive functional, performance or stress tests. A smoke test is also referred to as a build verification test or a sanity test.

***Smoke Test***: A “quick and dirty” initial test, to make sure the system can operate and process at least a minimal work load. The term “smoke test” comes from the saying by hardware developers: “Power it up and see if any smoke comes out.” In complex systems where the build or integration process is not straightforward, a smoke test is a quick post-build test to ensure that all the components have

successfully been integrated together. Its purpose is to provide assurance of the build process before proceeding to more comprehensive functional, performance or stress tests. A smoke test is also referred to as a build verification test or a sanity test. A phrase used to describe a subset of tests, typically limited in number, that can be run against each software build to determine whether the software has regressed in form or function since a previous build. Synonyms: build validation test, build verification test, build acceptance test, build regression test and sanity check.
Soak test. Running a system at high load for a prolonged period of time. For example, running several times more transactions in an entire day (or night) than would be expected in a busy day, to identify and performance problems that appear after a large number of transactions have been executed.
***Soap-Opera Test*** Soap opera testing takes “normal” user scenarios which the feature testers have already embedded in their feature test cases, and deliberately exaggerates them in order to heighten the demands on the system. A technique for defining test scenarios by reasoning about dramatic and exaggerated usage scenarios. Like a soap opera on television, these scenarios reflect "real life", but are condensed and exaggerated to depict dramatic instances of system use. When defined in collaboration with experienced users, soap operas help to test many functional aspects of a system quickly and because they are not related directly to either the systems formal specifications, or to the systems features they have a high rate of success in revealing important yet often unanticipated problems.
Software component. See module, component.
Software end game. The last 10% to 25% of as software project – usually activity is hectic as deadline pressures are intense.
Software engineering. The field of software development and maintenance.
Software fault injection. See error seeding, bebugging.
Software interrupt. See interrupt.
Software maintenance. See maintenance.
Software metrics. Quantitative measures of software processes and products. See measurement, metric.
Software package test. Testing an installed package to ensure that it works correctly, and interfaces and fits into the technical and business environment.
Software package. A vendor-supplied system which is ready to install, or to adapt the the specific user’s needs and then install. Also called third-party software, COTS and commercial off-the-shelf software.
Software plan. (1) A plan for a software development, maintenance or deployment project. (2) The high-level software design architecture for a system.
Software process. A set of activities, methods, practices, and transformations that people use to develop and maintain software and associated products (e.g., project plans, design documents, code, test cases, and user manuals).
Software quality analyst (SQA). Title of someone who performs testing and quality activities. See tester.
Software quality assurance (SQA). See quality assurance.
Software quality engineer (SQE). Title of someone who performs testing and quality activities. See tester.
Software quality. See quality.
Software reliability engineering. Method for predicting long-term software reliability based on short-duration test results.
Software requirements specification. A document that describes the data, functional and behavioral requirements, constraints, and validation requirements for software.
Software test. A set of activities conducted with the intent of finding errors in software.
Software. The programmed instructions which direct the behavior of computer hardware.
SPA. Software process assessment. See CMM, ISO.
SPC. Statistical process control.
***Spike and Bounce Test*** Spike testing utilizes an intense spike in the work load, usually for a very short duration, to determine how the system handles abrupt increases in demand. A variation of spike

testing is to follow the spike with a bounce down to a very low load level, and then continue repeating the up and down pattern. This tests whether the system can respond to rapid significant changes and re-direct the use of its resources, for example, through load balancing. An example of a spike test is to transmit a "send to all" e-mail, where we send a large file attachment to a large group of users at the same time. For example, we could use one of the PowerPoint presentations from marketing which incorporates video and audio clips (those marketing guys sure are long-winded). We send the file in uncompressed form and encrypt it so that every copy has to be decoded at the point of reception, and send it to everyone in marketing. This transmitted file should be reasonably large. In my experience, the three biggest hassles with spike-and-bounce testing are (a) determining how high the test spikes should be, (b) determining how far apart the test spikes should be on average (they should not be evenly spaced), and (c) determining what bottlenecks or throttles in the test environment may prevent the spikes from reaching their desired heights. Spike tests are tests that use real-world distributions and user communities, but under extremely fast ramp up and ramp down times. It is common to execute stress tests that ramp up to 100% or 150% of expected peak user-load in a matter of minutes rather than the hour normally allotted for ramp up. These tests are generally only executed after several rounds of tuning. If all components of the system continue to function normally, no matter how slowly, a Spike Test passes.
Spiral development. See iterative development.
Spiral model. An iterative or evolutionary software development method. See RAD, iterative development, prototyping, extreme programming, agile methods.
Spreadsheet test. Validating that the spreadsheet model and implementation are accurate.
SQA. See software quality assurance, quality assurance.
SQE. Software Quality Engineering.
SQL (Structured Query Language) test. Testing inquiry calls, data retrievals and updates to a relational database and made using an SQL-compatible interface or data management tool.
SRAM. Static random-access memory. A type of RAM which retains its data content as long as the system is powered on. Contrast with DRAM, where the memory must be continually refreshed (re-written) even if there is no change in the data contents.
SRE. See software reliability engineering.
***Stability Test*** Stability tests exercise a system at and beyond the worse expected demand it is likely to face. The majority of critical deficiencies in the system will have already been identified during the execution of load/performance tests, so tests deal more with assessing the impact on performance and functionality under a heavy or unreasonable load. Stability scenarios will also identify many other system bottlenecks not previously noticed, which may in fact be partially responsible for earlier identified problems. A stability test is a type of load test that assesses how dependable and robust a Web application is, rather than its responsiveness or throughput. This kind of test places a consistent work load on the application being tested for a considerable period of time.
Stability. Do existing features, which have not been changed, perform the same as they did in earlier versions of the system, after an unrelated feature has been changed?
Stack. An area of memory that contains a last-in-first-out queue of storage for parameters, automatic variables, return addresses, and other information that must be maintained across function calls. In multitasking situations, each task generally has its own stack.
***Staging Server*** A server used as a temporary stage to test new or revised Web pages before they are made live.
***Standard Benchmark Test*** This type of testing uses a standard work load rather than a user-specific one. It is common when it is difficult to discern how a system actually will be used (or is already being used), so comparability replaces realism. We measure performance when the system is under this standard load (also called a benchmark in some circles), instead of under a load which is based on the behavior patterns of a group of users. Standard benchmark testing is often used in comparing vendors'

products, such as database management systems (DBMS), which many different users can install and use in many different ways. The standard benchmark provides a means to compare the performance of the competing products under the same load and with the same resources. For this reason, testers sometimes call it product comparability testing. Industry associations such as the Transaction Processing Performance Council (TPC) provide benchmarks, as do the larger software, hardware and network services vendors. SAP, for example, provides a benchmark for its enterprise resource planning (ERP) software. The Benchmarking Institute of Gilroy, California, maintains a list of publicly available benchmarks. In many vendors, performance testing is a process of running industry benchmarks with the optimal configuration of a system and its environment, in order to obtain the best possible performance numbers for marketing purposes. In addition, most vendors of highly configurable products offer their products for sale in a limited number of pre-set configurations, each with a recommended standard support infrastructure for that configuration. Any one of these configurations can provide a starting point from which an individual customer tunes and optimizes its own system installation, but the vendor tests the system's performance and robustness using only the vendor's standard configurations.

***Standard Deviation*** A measure of dispersion which tells how values are dispersed from the mean (average) in a set of data. It is calculated by taking the square root of the variance.

Standard. (1) Measure. (2) Level of acceptability to be accomplished. (3) Guideline. (4) Mandatory practice.

Start-up code. Initial small software or firmware module, used to start the computer and load the operating system. See bootstrap, IPL.

***State Transition Diagram*** A graphical notation which shows the states of a system and the transitions among those states. See state-based testing.

State-transition diagram test. A technique to develop test cases based on the states that a system can have. The idea is to test the triggers or stimuli that cause a transition from one condition (the initial state) to another (the final state). Popular with engineers, who usually already are familiar with this approach from engineering designs. These diagrams are also called finite-state diagrams.

Static analysis. Analysis of program code carried out without executing the program.

Static analyzer. A tool that carries out static analyses.

Static test. A test that does not require actual computer execution, e.g., a peer review or desk checking.

Statistical process control. SPC is a quality technique in manufacturing operations, which uses control charts to plot the behavior of repeatable processes and to identify variations which are not successful.

Statistical quality assurance. Process improvement methods which use statistically valid measurements of products and processes, and statistical analysis of these measurements.

***Statistical Sampling*** Use of statistical techniques to determine the ideal size of a sample for testing, i.e., how many test cases are required to prove or disprove a hypothesis about what defects are present in an entire population.

Statistical sampling. Use of statistical techniques to determine the ideal size of a sample for testing, i.e., how many test cases are required to prove or disprove a hypothesis about what defects are present in an entire population.

Statistical testing. Testing which uses statistical methods in various ways, such as sampling, design of experiments and orthogonal arrays.

STD. (1) Standard – a common abbreviation on the U.S. government and especially in the military. (2) State transition diagram.

***Steady-State Work Load*** In a load test, a steady-state work load is a work load type where a consistent number of virtual users are run for the entire duration of the test. The use of a steady-state work load provides exact response time and throughput information for a given number of users. This type of work load is often employed for stress and stability tests.

Step-wise refinement. A top-down iterative method of requirements definition, design and programming.

Also called decomposition or partitioning.
Storage test. Testing that verifies the program under test stores data files in the correct directories and that it reserves sufficient space to prevent unexpected termination resulting from lack of space.
Strategy. Overall approach to getting something done. The set of decisions that have a significant potential to make a major impact on the success of the undertaking.
***Stress Test*** A stress test is one which deliberately stresses a system by pushing it beyond its specified limits. The idea is to impose an unreasonable load on the system, an overload, without providing the resources which the system needs to process that load. In a stress test, one or more of the system resources, such as the processor, memory, or database I/O access channel, often "maxes out" and reaches saturation. (Practically, saturation can happen at less than 100% of the theoretical usable amount of the resource, for many reasons.) This means that the testware (the test environment, test tools, etc.) must be sufficiently robust to support the stress test. We do not want the testware to fail before we have been able to adequately stress the system. Many bugs found in stress testing are feature bugs which we cannot see with normal loads but are triggered under stress. This can lead to confusion about the difference between a feature bug and a stress bug. Some testers prize stress testing because it is so fruitful in finding bugs. Others think it is dangerous because it misdirects projects to fix irrelevant bugs. Stress testing often finds many bugs, and fixing these bugs leads to significant delays in the system delivery, which in turn leads to resistance to fixing the bugs. If we find a bug with a test case or in a test environment which we can't connect to actual use, people are likely to dismiss it with comments like: "The users couldn't do that", ".. wouldn't do that" or "... shouldn't do that." Test application under a load for a period of time to discover the ability of the application to handle work load. Most commonly collects various performance related measurements based on tests that model varying loads and activities that are more "stressful" than the application is expected to encounter when delivered to real users. Sub categories may include: spike testing (short burst of extreme load) extreme load testing (load test with "too many" users) hammer testing (hit it with everything you've got, often with no delays) stress testing involves subjecting the product to heavy loads or stresses. Unlike volume testing, a heavy stress test is a peak volume of data encountered over a short span of time. If a product can only handle 50 simultaneous users on the system, then exercise the product with 51, 100, 200, etc users. Any testing involving challenging inputs or environments; where challenging means at, near, or exceeding the actual operating limits of the product under test (not merely its stated limits). Stress testing is done for the purpose of understanding how reliability and performance degrades over time as the product is "stressed" by such challenging circumstances. Good stress testing requires that the test designer analyze the potential vulnerability of the product to stress, so that the product is indeed challenged in every relevant manner and dimension.
***Stress, Robustness and Reliability*** Although stress, robustness and reliability are similar, the differences among them mean that we test them in related but different ways. We stress a system when we place a load on it which exceeds its planned capacity. This overload may cause the system to fail, and it is the focus of stress testing. Systems can fail in many ways, not just from overloading. We define the robustness of a system by its ability to recover from problems; its survivability. Robustness testing tries to make a system fail, so we can observe what happens and whether it recovers. Robustness testing includes stress testing but is broader, since there are many ways in which a system can fail as well as from overloading.
String test. A test performed on small strings of components which are connected together in their natural linkages.
Structural test. Testing based on an analysis of internal workings and structure of a piece of software. See also white box test.
Structured design. A design method which focuses on modularity.
Structured programming. A programming method which emphasizes modularity and which prohibits

jumps outside of modules.
***Structured Test*** A test which uses defined processes and analytical test techniques.
Structured testing. Testing which uses defined processes and analytical techniques such as equivalence partitioning.
STSC. Software Technology Support Center, a part of the U.S. Air Force with useful materials available to testers.
Sub-test. Component or part of a test.
Supportability. The customer support people can adequately support this system, e.g., they have sufficient aids to be able to troubleshoot telephone calls from customers.
Survivability. In systems and networks, the ability to recover and continue in operation, even with a degraded level of service. What is the likely survivability of the system after disasters, even if it continues operation in a degraded mode with limited capabilities? This is an important characteristic of open networks, such as the Internet, where equipment is continually added and removed, users are continuously connecting and disconnecting, and where the overall network is expected to continue operating even when parts of the network are down.
Sustainability. Types of sustainability issues include data integrity, auditability, maintainability. re-usability. avoidance of side effects, and supportability.
SUT. System under test.
Symptom. An indication that something undesirable has occurred or may occur.
Synchronization. (1) Coordination between co-dependent events. (2) The process by which an automated testing tool waits until an event occurs.
***Synchronization Test*** This type of testing attempts to stress a system by causing timing problems and out-of-synch process. These are also called race conditions. Systems often have inadvertent and unrecognized assumptions built into them, about the expected sequence of events of the expected timing of events. Let's say that the system assumes if event A always precedes event B. What happens if this assumption is not met? Let's say an event happens later – or earlier – than anticipated. Does the system time-out (and is not supposed to) because the late event? Does the early event go unnoticed? The aim of synchronization test cases is to answer these questions.
Syntax-driven test. Technique used for testing compilers and operating systems. Based on the syntax of the language or input commands, build a command-line generator or a table of alternative input commands, including both valid and invalid values, and use these to drive the testing.
System design. See design.
System development life cycle (SDLC).
System integration.
System operational risk. A significant possible loss which may be caused by using the system.
System release.
System requirement. See requirement.
System risk. A significant chance of a loss associated with a particular system.
***System Test*** (1) Highest level of application functionality testing performed by the systems group, or by a combined systems and user group, and usually on the completely assembled product. (2) Any test which is performed on the fully integrated system. Synonym: functional, feature or behavioral test. Testing that attempts to discover defects that are properties of the entire system rather than of its individual components.
***Table & Array Testing***. Tests of the table look-up and maintenance mechanisms, including security, and tests for the data integrity of the table entries. May include tests like trying to delete a non-existent record, detection of duplicate entries, ability to reject invalid data (e.g., changing a customer name to all blanks).
Target hardware. The platform where the software will run in live operation, as opposed to the

development or test platform. See host; environment.
Task. (1) A work activity, which may be part of a project or stand-alone. (2) A piece of computational work which is executed by a computer processor. Unlike processes, tasks share a common memory space and must be careful to avoid overwriting each other's code and data.
TBD. To be determined. Often found in test plans and other planning documents. When in doubt, the testers can insert “TBD” to help fill the test plan. (This comment is meant as a joke in dubious taste.)
Tcl. Tool command language. Popular C-based test language.
Technical compliance. Does the system comply with the relevant technical standards, e.g., operating system interface standards, system documentation standards? Does the system or product support an open, plug-and-play architecture? Technical compliance may overlap – or conflict -- with compatibility.
Technical risk. A potential technical problems which cause a project or system to fail.
Technology. Applied science, engineering.
Test automation. Testing employing software tools which execute tests without manual intervention. Can be applied in GUI, performance, API, etc. testing. The use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.
Test bed. The test environment. Alternative definition: a restful support mechanism for tired testers (this idea is conceptual, never implemented). A test bed is an execution environment configured for testing. May consist of specific hardware, OS, network topology, configuration of the product under test, other application or system software, etc. The Test Plan for a project should enumerated the test beds(s) to be used.
Test case A particular condition or situation to test.
Test case design. The process of analyzing the decision logic and characteristics of a feature or a set of features, and deriving a set of test cases to exercise that feature. Test case design is based on the requirements, expected behavior and constraints of a system or a software component.
Test case. An executable test of one particular situation or condition, with a distinct or unique set of input data values and initial conditions (i.e., the environment), and with a corresponding pre-defined expected result of the test. Synonyms: test script, test condition, test scenario, test assertion, test specification.
*Test Case / Test Data Library* This library contains the test suites, test cases and components from which the test cases are built. May include procedural components (e.g., re-usable test software), and data components (e.g., tables of input data values and expected results).
*Test Case Configurator* These tools tune or configure generic test cases to fit specific versions of the system under test, or specific test environments (e.g., different platforms). The only tools available to automatically configure test environments are “home grown”; no tools are known to be commercially available.
*Test Case Generator* These tools generate test cases directly from the specifications (automated test planning). Based on the functional specifications, the tools use techniques like cause-effect graphing to identify what to test. The tools generate test cases that are intended to be ready for execution.
*Test Case Selector* These tools extract subsets of test cases from test case libraries, according to a set of selection criteria.
*Test Case Version Control* Version control on the test case library.
*Test Data Base Loader* In data-rich systems, the amount of test data can be voluminous (hundreds of gigabytes in the case of Experian, formerly known as TRW Credit Data). These tools maintain and load test data bases, e.g., data bases of customers and orders.
*Test Data Generator* These tools automatically generate sets of test data, in the form of test transactions or test master data bases.
*Test Data Manager* These tools organize and manage test data (test cases, test results, etc.).
Test documentation. Written record of test plans, test cases and test results.

# *Appendix A: Basic Definitions and Concepts*

*Test Driver* The test driver is a tool which drives the testing process. Typically, this tool interfaces with the system under test (SUT), either generates or extracts test data, initiates the test process, and undertakes a series of steps to feed in the test data. Examples include call generators and TCP/IP packet drivers. Also called the test engine. In commercially available tools, the test driver is usually combined with a test results capture tool.

***Test Duration*** How long should a performance test run? Five seconds, five minutes or five hours? With the fast processing speeds available today, more testing can be done more rapidly, and extended test durations arguably have become less important. Because an entire project may have to wait for hours or days while a long-duration test runs, it is a good idea to review the bug reports to see what new information, if any, the second hour's worth of testing adds beyond the first hour, the third hour beyond the second, and so on. In addition, the field of software reliability engineering does not require inordinately long test times in order to evaluate system reliability. And if we are preoccupied with the mechanics of just running the test, we don't have much time to think and strategize. The minimum test duration depends on these factors: The time needed to initiate a process, including checking the test pre-conditions and starting test tools, ramp up and reach a reasonably steady state of operation. The time needed to gather sufficient data to satisfy sample size requirements or targets for data sufficiency. For example, in some test runs the work load varies in a roughly periodic pattern during the performance measurement, to reflect the typical pattern of how the load ebbs and flows over time. This provides a more realistic picture of the system's performance characteristics than testing with an even load, but the test duration must be long enough to capture at least one full period of data. Whether the test objectives include finding so-called slow burning fuse defects, such as slow memory leaks, which can take days to surface.

***Test Environment*** The support infrastructure which is needed to operate the system in test mode, and the facilities needed to test it, including test libraries & data files, procedures, tools, etc. Synonyms: test infrastructure, test bed. The test environment is the hardware and software environment in which tests will be run, and any other software with which the software under test interacts when under test including stubs and test drivers.

*Test Environment Management Tool* Any tool used to help manage the environment, e.g., configuration management in the test environment.

Test factor. Any characteristic which can vary during live operation, such as the sizes of the semiconductor memory and the hard disk installed in a personal computer, and which may affect the test results.

Test factor analysis. The process of identifying the test factor combinations which need to be tested. Each test factor can have several possible options (e.g., in the case of the hard disk, the size options could include 1 GB, 5 GB, 10 GB and 25 GB). All combinations of test factors are usually much too numerous to test, so test factor analysis identifies which combinations of test factors actually need to be tested, under various assumptions about the relationships among these test factors. See orthogonal arrays.

Test harness. Support infrastructure required to generate, format and feed in the input data required for a particular test; and also to capture, interpret and log or present the results. See test environment.

Test infrastructure. See test environment.

Test log. The repository of the results of each test case executed in the test project.

Test pattern. A re-usable test procedure, template or checklist.

***Test Plan*** A plan to test a system, product or subsystem A document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.

***Test Procedure*** The series of tasks or steps which must be performed in order to execute a test case or for another test process. The procedural aspect of a given test, usually a set of detailed instructions for

the setup and step-by-step execution of one or more given test cases. The test procedure is captured in both test scenarios and test scripts.
Test requirement. A brief statement of a test need, to serve as the basis for test case design.
*Test Results Capture Tool* This tool interfaces with the system under test, and captures and logs the test results. This means that the capture tool must be fast enough to capture transient behavior. The test results capture tool may or may not be invasive, i.e., it may interface only externally and capture the end results, or it may use probes to capture information about the internal state of the system during testing.
*Test Results Data Manager* Manages the test result log, which is usually voluminous. Provides capabilities to store test results,, cross-reference them to test cases and SUT versions, browse and extract test results, and analyze patterns. Usually also perform version control on the test results.
*Test Results Log* The test log is simply a record of the results of each test case that has been executed. While the test log appears humble and ubiquitous, and is seldom seen outside the test team's work area, it plays an important role in testing.
Test script. (1) The generic condition which needs to be tested, or alternatively the family of all test cases which test that condition (i.e., the test cases exercise the same condition, and are identical except for variations in their input data values and expected results). The test script is often called a test condition, a test scenario, a test procedure or even a test case, as there is much use of confusing terminology in this area. (2) A document that specifies for every test and in a test suite: object to be tested, test requirement, initial state, inputs, expected outcome, and validation criteria. (3) Commonly used to refer to the instructions for a particular test that will be carried out by an automated test tool.
Test specification. A document specifying the test approach for a software feature or combination or features and the inputs, predicted results and execution conditions for the associated tests.
*Test Status Reporting Tool* These tools extract test results, and automatically compile reports and graphs of test project status and trends.
Test suite. A group of related test cases. Synonyms: test cluster, test group
Test tool. Computer software or hardware used in the testing of a system, a component of the system, or its documentation.
***Testability*** Can the system be validated, i.e., shown to work in an acceptable manner? Testability is the ability to perform trustworthy and cost-effective testing and degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.
Test-driven development. Testing methodology associated with Agile Programming in which every chunk of code is covered by unit tests, which must all pass all the time, in an effort to eliminate unit-level and regression bugs during development. Practitioners of TDD write a lot of tests, i.e. an equal number of lines of test code to the size of the production code.
Tester. A person who tests a system or component. "The guy who takes away the punch bowl just when the party is getting good." (To slightly misquote Alan Greenspan, chairman of the Federal Reserve Board.)
Test-first design. Test-first design is one of the mandatory practices of Extreme Programming (XP).It requires that programmers do not write any production code until they have first written a unit test.
Testing. An organized process of: (1) validating that a system works as expected, (2) identifying discrepancies, i.e., the variances between the actual vs. the expected results from the use of a system, and (c) providing information about the state of the system.
Testware. The name for the support tools, test case libraries and test databases, test procedures, training, and the test lab support organization. See test environment, test harness.
***Think Time*** How long a user spends viewing a page before clicking the next link.
Third-party software.
Thrash. This happens when a processor becomes unstable and sends most of its time switching among

tasks, rather than working on those tasks.

***Thrashing*** Thrashing is computer activity that makes little or no progress, usually because memory or other resources have become exhausted or too limited to perform needed operations. When this happens, a pattern typically develops in which a request is made of the operating system by a process or program, the operating system tries to find resources by taking them from some other process, which in turn makes new requests that can't be satisfied. In a virtual storage system (an operating system that manages its logical storage or memory in units called pages), thrashing is a condition in which excessive paging operations are taking place. A system that is thrashing can be perceived as either a very slow system or one that has come to a halt.

Thread Testing. A variation of top-down testing where the progressive integration of components follows the implementation of subsets of the requirements, as opposed to the integration of components by successively lower levels.

Thread. A computational task. In multi-threaded systems, multiple threads run concurrently, and share the same software components and common data.

Threat analysis. See hazard analysis.

Threat. See risk.

***Throughput*** The amount of information moved in a given amount of time. Throughputs are usually measured in Kbps, Mbps, or Gbps, but may be measured in transactions per second. The throughput rate is the average rate at which an application processes requests. Throughput rates are an important factor in the performance of an application. Common throughput goals include that the application be able to handle a specific number of requests per hour and that the processing time for a high percentage of requests be below a specific time limit.

***Throughput Test*** Throughput testing measures how much traffic passes through a system within a specified period of time and under a specified load. This test load may be light, average, heavy or vary over time. We can measure throughput in megabits per second, events (database queries, requests or transactions) per second, or another metric. The selection of the units of measure for throughput can influence the test results. For example, let’s say that the test objective is to rank a group of competing servers from best to worst, in terms of throughput. Their rankings may be different if we measure the throughput in megabits per second than in events per second. In other words, a server which is the best of the group when we measure the servers’ performance in megabits per second could look the worst in terms of the number of database queries processed per second. (Can you identify possible reasons for this seemingly illogical situation? Hint: a high number of queries will not translate into a high number of megabits if the average query length is short. Or if we count only the directly useful data in the queries, not the overhead such as headers and trailers wrapped around the core of useful data, and the queries contain large amounts of inefficient overhead in the form of extra bits.) The recorded throughput also depends on where in a system we count the bits or the events – the volumes of events usually are not the same at each internal point. We can count the throughput in a network as the amount of traffic which originates from, is received at, or passes an internal point within a given period of time. In a simple situation where one specific input triggers each output, the count of the output traffic received at the destination is exactly the same as the count of the input traffic. But this ratio can be less than 1 to 1, within a given period of time, if there are bottlenecks and inefficiencies within the system, or can be higher than 1 to 1 if a single stimulus triggers the broadcast of multiple messages. There also may be questions about what to count and how to count it. For example, let’s say that a Web server has an ongoing, low-level flow of administrative management messages and error messages, as well as the “real” traffic, namely the requests from visitors to the Web site which this server supports. The measurers will need to decide which of these traffic categories to include in their throughput counts.

TickIt. A British method or assessing the effectiveness of quality practices. Similar to CMM and ISO certification.

# *Appendix A: Basic Definitions and Concepts*

***Time of Peak Demand*** (e.g., the Peak-Demand Hour or Peak-Demand Minute) The timing of the peak demand. The peak-demand hour in a time period (e.g., a day or week), is the hour with the highest volume during that period. It is also referred to simply as the peak hour. There can be more than one time of peak demand within a duration, if two or more intervals have the same volume. The peak-demand hour in a typical week will occur at the same time as the peak hour of one day during the week, but may or may not occur at the same time as the peak hours in the other days of that week.

Timeliness. Are the developers able to deliver or modify the system within the timeframe when it is needed?

TLA. Three letter acronym. A key tool in snowing managers.

Tolerance. (1) The allowable margin for imprecision before is considered to be unacceptable.

Top-down test. An approach to integration testing where the component at the top of the component hierarchy is tested first, with lower level components being simulated by stubs. Tested components are then used to test lower level components. The process is repeated until the lowest level components have been tested.

Total quality management (TQM). (1) A top-down, organization-wide effort to improve quality and customer satisfaction through publicizing the importance of quality, training, motivation and focus on quality improvement opportunities. (2) A company commitment to develop a process that achieves high quality product and customer satisfaction.

TQM. Total quality management.

Trace. A record of the execution path through a system or software component, usually in the form of a list in time sequence of the lines of source instructions which were executed.

Traceability Matrix. A document showing the relationship between test requirements and test cases.

Traceability. The ability to link test cases back to the specific system requirement(s), feature(s), configuration(s) or constraint(s) being tested.

Transaction flow test. A technique to develop test cases from transaction flows. Identify the inputs to a system, predict the end results of each system (outputs and updates to permanently stored data), and test to verify that these results do in fact occur.

***Transaction*** In Internet and Intranet applications, a transaction is a sequence of Web application accesses, interspersed with user interaction, and having a certain logical unity. For example, a transaction might consist of a user visiting an e-business application, browsing its contents, ordering a product, and then leaving the application.

Trap. An interruption of processing which is triggered by the internal hardware, for example, the processor might call a trap if an illegal opcode is encountered during execution. See interrupt.

TRR. Test readiness review, to ensure the test entry criteria have been met.

***Typical Hour*** An hour with average traffic and the demand generated by that traffic.

UAT. User acceptance test.

UML. Unified Modeling Language.

Unified Modeling Language. A standard set of notations and conventions for documenting use cases.

***Unit or Component Test*** A stand-alone test of a low-level component, module or subprogram test.

Unit or component test. A stand-alone test of a low-level component, module or subprogram. It is the foundation for later levels of testing. Usually this is a personal activity of the software engineer or author, who should take accountability for his or her own work.

Unit. Separately testable element, specified in the design of software.

Upgradeability. Are there planned future upgrade paths, i.e., as the use of the system grows, can the software, databases, hardware and networks be upgraded reasonably easily to support this growth?

Upstream/downstream test. This technique applies where there is a series or sequence of systems, which feed data from an upstream system to a downstream one. The technique focuses on (a) which upstream data feeds must be tested after a change to a system, and (b) which downstream system(s)

and data feeds are likely to be impacted by that same change.

***Usability Test*** This type of testing involves the usability of the system or is the product designed well enough for the consumer who will be making daily use of the application. This type of testing is often performed by beta testers and a user acceptance testing team to determine the usability rating.

Usability test. (1) Checking the product's user friendliness, which is it's ease of use and ease of learning for the typical user. (2) A test which addresses these questions: Is the system easy to use and to learn? Does it improve customer satisfaction, productivity, business reliability and user morale?

Usability. Can the typical user run the system productively and without aggravation? Is it easy to learn and use? Usability considerations also include ease of training, availability of on-line help built into the system, availability of work-arounds, flexibility to configure or customize the system to individual user needs, and ability to meet divergent user needs.

***Usage-Based Test*** Realism is the key success criteria for this type of testing. We expect the performance measured in testing to predict closely the performance which the users will experience in live operation, by a particular user group which is working in a particular technical environment. We take care to ensure that the ways we exercise the system in testing are the same as it will encounter in live operation, and the test environment closely mimics the infrastructure which will be used to support the system in operation. With reality-based testing, we move away from finding performance and robustness defects for the sake of finding defects, that is, problems which can be created in the test lab but which the users are unlikely to ever encounter. Instead, the lab testing helps find defects which otherwise could have consequences for the users.

***Use Case*** A model of how users interact with a system.

Use case. A use case describes the interaction between a system user and the system which fulfills a particular need of the user. An example of an interaction which can be described by a use case is the act of withdrawing money from an automated teller machine.

User acceptance test (UAT). See acceptance test.

***User Delay*** How long a user spends viewing a page before clicking the next link.

***User Experience*** Any measurement that relates to what a user actually sees (experiences) when they use the system. User experience generally refers to end-to-end response time. User experience is about usability, which has a broader scope than user interface (UI). It includes the application's look-and-feel; not just what the user-interface provides from a functional point-of-view. However, there is a component if user experience that is concerned with UI. Encompasses all aspects of the end-user's interaction with the company, its services, and its products. The first requirement for an exemplary user experience is to meet the exact needs of the customer, without fuss or bother. Next comes simplicity and elegance that produce products that are a joy to own, a joy to use. True user experience goes far beyond giving customers what they say they want, or providing checklist features. In order to achieve high-quality user experience in a company's offerings there must be a seamless merging of the services of multiple disciplines, including engineering, marketing, graphical and industrial design, and interface design.?

***User Group*** In a load test, a user group is a number of virtual users regarded as a single unit. The virtual users in a user group all carry out similar transactions with the Web application under test.

User In the context of e-business, a user is a person who accesses the World Wide Web by means of a computer and a Web browser. Users are the customers, or potential customers, of e-business applications. The Web behavior of real users is simulated in load testing through the use of virtual users.

***User Interface (UI)*** The interface through which a person works with a system.

***User Scenario Test*** User scenario test. A technique to develop test cases based on users' work flows and their typical utilization of the system to perform their jobs or business functions, and use these as the basis for testing. Useful for testing external parts of systems that have user visibility, such as the web pages or GUI front ends of systems.

# *Appendix A: Basic Definitions and Concepts*

User. A person who uses the software or the software-based product such as an embedded device.
V&V. Independent Verification and Validation (IV&V).
***Validation*** The process of determining the system does the right things, i.e., complies with the stated requirements; performs the functions for which it is intended; meets the organization's goals and the users' needs. Validation is the process of comparing the intermediate and final deliverables during system development back to the original overall project objectives. The process of evaluating software at the end of the software development process to ensure compliance with software requirements. The techniques for validation are testing, inspection and reviewing.
Vector. A list, column or one-dimensional table.
***Verification*** The process of determining the system does things right, i.e., was built following reasonable design principles; provides a consistent user interface; performs the selected functions in the correct manner; accesses the right data, and has undergone an appropriate series of reviews. Verification is the review of interim work steps and interim deliverables to ensure they are acceptable, and is the process of comparing the deliverables of an intermediate phase to the conditions imposed at the beginning of that phase. Verification is the process of determining whether of not the products of a given phase of the software development cycle meet the implementation steps and can be traced to the incoming objectives established during the previous phase. The techniques for verification are testing, inspection and reviewing.
Version control. The mechanism(s) for controlling changes to systems and assessing their current status. In more complex situations, this control is entitled "*configuration management*" and extends to the system environment, documentation, test beds, test results, interfaces, and other system-related areas.
*Version Control Tool* Since most systems being tested could be set to many different configurations in live operation, these tools set, control and change the configuration as needed for testing. They also record the specifics of each test configuration, so that the configurations can be cross-referenced to the test results. These tools may be the same as -- or different to -- the tools used to manage the configuration of the test environment outside the system being tested.
***View or Page View*** A single user-initiated event, activity or transaction; typically a request by a visitor to download a web page in order to view it. When a visitor views a web page containing four graphs, the number of hits is five – one for the page itself and for each of the graphics files. For this reason, web statistics measured in views are a more useful metric than the numbers of hits.
Virtual machine. A computer simulated in software on a different hardware platform.
***Virtual User or Virtual Tester*** A user simulated by an automated testing tool. Performance tests are run with virtual users. A virtual user is a simulation of a real user conducting transactions on an application. During a performance test, many virtual users can be run from one computer. Load tests run a number of virtual users. A virtual user is a program that acts just like a real user would in making requests to a Web application. During a load test, a considerable number of virtual users can be run on one computer, in this context called a driver machine.
Virus test. Testing that virus controls are adequate.
Visualization. A means to allow users to navigate and search "naturally" through unfamiliar information landscapes and to manage large-scale and complex multimedia data sets.
***Visit or Session*** The series of individual events or activities (usually the externally visible interactions between one user and the system), which occur from the time when the user logs on to the system until the same user logs off.
***Visitor or User*** The person who visits the web site (and thus initiates a session).
V-model. See waterfall approach.
***Volume Test*** This type of testing usually combines functional testing, which is conducted with sizeable volumes of test transactions, with performance and robustness measurement -- getting “two birds with

one stone". Volume testing is sometimes called parallel testing (see below), when we run the same volume of test transactions in before-and-after comparisons, before and after a system modification is made. Volume and parallel testing are most popular in mainframe testing because their logistics are relatively straightforward in that environment. Volume testing is sometimes called work flow testing also.
Any test focused on "how much" instead of "how fast". Often related to database testing. The distinction between "volume" and "load" is that volume focuses on high volume and does not need to represent "real" usage. Subject the product to heavy volumes of data. This form of testing shows that the application can not handle the volume of data specified in its objectives. For instance, batch process designed to import data on a nightly bases may have a limit to the amount of information it can handle before failing, so give the system a larger amount than this to see how it responds.
Volume test. Testing by processing a significantly large volume of transactions, which often are copies of live data. Volume testing in some circumstances also is called parallel testing. Alternatively, the term "volume testing" is sometimes also used to mean load or stress testing, which are different from before-and-after comparison parallel testing. Volume Testing includes testing which confirms that any values that may become large over time (such as accumulated counts, logs, and data files), can be accommodated by the program and will not cause the program to stop working or degrade its operation in any manner.
Vulnerability. See risk.
Walkthrough. A review of requirements, designs or code characterized by the author of the material under review guiding the progression of the review.
***Watchdog Timer*** A hardware timer that is periodically re-set by software, and then used to count down as the execution continues. If the software crashes or otherwise experiences problems, the watchdog timer will expire (count down to zero), and the system is re-set automatically.
Waterfall model. Traditional system development process which follows a series of phases, such as requirements definition, design and coding. See: V-model, system development life cycle (SDLC).
WBS. See work breakdown structure; functional decomposition.
Weakness. See risk.
Web-based application test. Testing applications which use the Web, such as Web pages and Web sites.
Whistle-blower. Someone who alerts the managers, regulators or authorities of a significant problem. May be accompanied by personal risk.
White box test. A low level, detailed test, based on the internal structure of a program or system rather than on the functional specifications. Synonyms: structural test, component test, code-based test, unit test.
***White-Box Test*** A low level, detailed test, based on the internal structure of a program or system rather than on the functional specifications. Syn. structural test, component test, code-based test, unit test. White box testing treats the system as a collection of many parts. During white box testing, many diagnostics may be run on the server(s), the network, and even on clients. This allows the cause(s) of the bottlenecks to be much identified more easily, and therefore addressed. White box testing can go as deep as individual lines of code, database query optimization, or memory management of segments of code.
Work breakdown structure. See functional decomposition.
***Work Load and Work Flow Test*** A work load is a mix of demands on a system. The term work flow is sometimes used to mean the same thing, or we simply call it the load. Depending on the situation, these demands can be events, transactions, queries, interrupts, or other demands, or a mix of these. We could claim that by this definition any mix of demands qualifies as a work load– and it does. The critical question is what the work load represents. Having the appropriate test work load (or more likely, set of related work loads) is crucial, because most system characteristics can vary significantly with the work load – performance, reliability, and so on. This means that the critical success factor for a work load is realism. If the work load used in testing is reasonably close to the work load of a particular user, then we

can say with confidence that the conclusions formed by testing in our lab are likely to apply to that user. If not, it is anybody's guess how well the lab findings predict what will happen in that user's environment.
The term "work load" test does not imply any particular set of test objectives or measurements (though internally an organization might use the term as a label for their own particular flavor of testing)
Depending on what we want to accomplish, a work load can be used in many different types of testing. For example, we could run a work load while we measure the system's performance (response time, throughput, availability, etc.) Or we could run a heavy work load to stress the system, attempt to overload it and see if it recovers from failures. Work loads are used in interoperability testing (checking if subsystems connect and communicate), configuration testing (checking if features are compatible across different system versions, and across different support hardware, databases and networks), and so on. A complicating fact is that many systems are highly configurable – they can be modified through changes in switch settings, patches and upgrades to behave differently, and can run with different variations of hardware, etc. Another complication is the huge number of users of a major system, such as an operating system (OS) from a major vendor, all of whom use the system in their own ways. We need a scheme to manage the numbers of variations of support environments, system settings and system uses. The common way to resolve being overwhelmed by the number of variations is a categorization model, which allows us to group together similar users by type of work load (usage pattern) and by type of support environment.

***Work Load Distribution*** A work load distribution is a representation of the functions performed by a user community on a system. For example, during the course of a day on a retail-based website, most users are shopping, some are searching for a specific product, some are finalizing purchases and checking out, while a single administrator may be updating product prices. A work load distribution is based on a percentage of users performing a specific function over a given period of time. Using the above example a work load distribution could be: shopping – 83%, searching - 5%, checking out – 10% and administration - 2%. In a load test, the work load is the total amount of simulated activity to which the Web application being tested is subjected. A work load consists of a designated number of virtual users who process a defined set of transactions in a specified time period. Work loads can be of various types, depending on the requirements of a particular load test. The principal types are steady-state work loads, increasing work loads, artificial work loads, and scenario-based work loads.

***Work Load Parameters*** Are characteristics of users' requests such as traffic intensity, system load, and resource consumption. Pay special attention to those work load parameters that will be affected by your alternative performance objectives as well as those that have a significant impact on performance - CPU utilization, total I/O operations, memory usage, page faults, and elapsed time.

Work plan. Description of the activities needed on a project to accomplish the project's goals.

Workaround. A fix for a problem, generally an expediency or compromise that is quick, inexpensive and temporary.

Workflow Testing. Scripted end-to-end testing which duplicates specific workflows which are expected to be utilized by the end-user.

***XML*** Extensible mark-up language.

XP. Extreme programming, a software development methodology.

# *Appendix B: Establishing Performance Requirements*

## APPENDIX B. ESTABLISHING PERFORMANCE REQUIREMENTS

### An Example of a System Performance Requirement

| **Transaction or Event** | **Throughput Load Volumes** (transactions per minute) | **Delivery Time or Response Time Goals** |
|---|---|---|
| 1. Request to download the table of contents page of the catalog. | 200 | 3 seconds or less, as measured at the visitors' workstations; 90% of the time or more |
| 2. E-mail notification of the new catalog's availability | 1,000 | 5 minutes or less; 90% of the time or more |
| 3. Request detail page of the catalog | 100 | 3 seconds or less; 90% of the time or more |
| 4. Conduct book search | 25 | 7 seconds or less; 90% of the time or more |
| 5. Place a new book order | 10 | 7 seconds or less; 90% of the time or more |
| 6. Request home page of the book club | 50 | 3 seconds or less, as measured at the visitors' workstations; 90% of the time or more |
| 7. Other background noise | 100 | No goals set |
| **Related Requirements** | **Aspects to Measure** | **Acceptable Thresholds** |
| 8. Resource utilization | Main processors, memory and network links | Utilization cannot exceed 65%, reserve capacity is not to fall below 35%. |
| 9. Error rates | Transactions listed above | Error rates not to exceed 1% for each transaction type |
| 9. Availability | Transactions listed above | Availability is covered by the error rate threshold (*) |

(*) Each test case is defined to contain one transaction as an input stimulus and the related set of outcomes for that stimulus. All attempts to execute a test case are counted. Transactions are counted as errors if they are not completed because the system is unavailable, and are included in the error rate computation.

## *Appendix B: Establishing Performance Requirements*

### The Process for Setting Requirements

First, review the business critical success factors and business objectives for the system, to the degree they have been stated. If you have no knowledge of what these are, you will need to ferret them out.

Second, answer these questions:

1. Where do we need to set goals? Which 5% of the performance envelope is most worth tracking and caring about? Put another way, which 5% of the system behaviors are in areas where the performance requirements are worth defining? It helps to address these questions:

- Which users have priority, in terms of receiving the best performance?
- Which types of work have priority?
- Which system uses are likely to cause the most user dissatisfaction or business losses if performance is poor?

2. How aggressive do the service levels need to be, i.e., the goals that are user-visible? (Service levels are goals that are meaningful to users. Internal disk utilization, for example, is not user-visible.) It helps to address these questions:

- Who are our main competitors? How do customers differentiate among us?
- In comparative evaluation with competitors, in which performance-related areas do we (a) need to beat the competitors, (b) meet them or (c) do not care as performance is irrelevant?
- If we are not in a competitive situation, what stated goals and unstated expectations are there for user productivity?

3. Under what demand conditions do these service levels need to be met?

- Under what expected load volumes do they need to be met?
- With what other concurrent demands (overheads and background noise) imposed on the shared resources?

- What rates of growth does the system need to accommodate?

4. With what resources (and therefore, at what cost) must the service levels be met?

- What types of resources does the system need to run on? (Do not bother to list these if they are obvious.)

- What quantities of these resources are normally available to the system?

- What are the desired norms for resource utilization?
    - Under typical load?
    - Under peak load?

- What are the desired norms for reserve or spare capacity, for each critical resource?
    - Under typical load?
    - Under peak load?

- In the trade-offs among the performance characteristics, such as response time and availability, which ones have high visibility and high risk and thus highest priority?

Third, document the performance requirements. Use a structure similar to the one in the earlier example.

**Setting Performance Requirements Early**

An application's service levels and thus its performance requirements ideally are defined early, during the requirements definition stage. This is not primarily a developer's task: users, customers or business managers need to establish what response times and availability are acceptable. It may be more useful to start by specifying what is unacceptable. Prior experience, competitors' actions and experimenting with prototypes can help with this definition.

Many organizations neglect to performance requirements before starting system implementation. Goals that are inadequately defined encourage poor design and coding, wasted efforts and re-work, systems that cannot be tuned and maintained, and significant risk of operational fiascos.

## *Appendix B: Establishing Performance Requirements*

### Mapping from the User's Perspective to the System Administrator's

While the primary performance requirements should be visible and meaningful to end users, the closer these requirements can be mapped to the physical architecture then the easier and the more objectively the performance can be monitored. If the operational environment is structured into tiers or layered (e.g., with an application layer, database layer and a network layer), we should try to define performance requirements at each layer so that each support team has its own set of performance targets to aim for. If this is not practical, the performance engineers still need to tune across all layers.

### Including a Performance Focus in the System Design

The architects and developers should be aware of the performance impact of different design choices, and deliver a blueprint which clearly identifies performance aspects of the architecture. Important steps to include in the design work:

- Analyze the design for shared and limited resources. For example, a network connection may be both a shared and a limited resource; a database table is a shared resource; while threads are a limited resource. These are the resources that probably will cost the most to fix later if they are not designed correctly.
- Estimate traffic, stored data volumes and load carrying capacities, and use these to determine if the design is feasible and identify the limitations of the system.
- Require performance predictions from the design.
- Include design experts familiar with the performance aspects of design choices in the external design reviews. If any significant third party products will be used -- like middleware or database products -- the product vendor should have performance experts who can validate the design and identify potential performance problems.
- Assess the system's scalability both for users and for data/object volumes; the amount of distribution possible for the application depending on the required level of messaging between distributed components; and the transaction mechanisms and modes (pessimistic, optimistic, required locks, durations of transactions and locks held).

## *Appendix B: Establishing Performance Requirements*

### Defining the Workloads

Performance measures like response time and resource utilization have complex relationships with the workloads, so setting goals without regard to the load is meaningless. A realistic and reasonably complete definition of the system's workloads is important for predicting or understanding its performance. A small change in workload can cause far more variation in the performance of a system than differences in processor clock speed or random access memory (RAM) size. The workload definition includes the types and arrival rates of requests to the system, plus any concurrent background noise (other demands on the same resources). Make sure you include the work that the system is doing "under the covers."

### Bypassing Load Calculations

We can bypass having to calculate the load, however, by setting availability percentages. Without knowing the average or peak loads, we can set a goal of let’s say 99% system availability or 99.999% (five nines), based on the level of customer service desired.

### Performance Objectives

After defining the workload that the system will have to process, you can choose performance criteria and set performance objectives based on those criteria. The main overall performance criteria of computer systems are response time and throughput.

We are primarily interested in four dimensions of resource utilization:

**Processor time**
: Processor cost of the workload

**Disk accesses**
: Rate at which the workload generates disk reads or writes

**Network traffic**
: Number of packets the workload generates and the number of bytes of data exchanged

**Real memory use**
: Amount of RAM the workload requires

# *Appendix C: The Initial Impact Assessment*

## APPENDIX C. THE INITIAL IMPACT ASSESSMENT

### Introduction

Performance testing, including measurement in a test lab, evaluation and prediction of live behavior, is one of the most expensive, system-implementation-delaying and resource-demanding types of system testing. The resources required include skilled performance testers, specialized tools and performance testing labs. Performance improvement projects also can be expensive and time-consuming, and draw on specialized resources to tune and fix the system.

An initial assessment provides an early filter to help determine where best to deploy the scarce testing resources. Since many activities can help ensure that the organization meets a system's performance goals, the follow-up actions considered in an impact assessment are broader than just performance testing. They can include predictive modeling, assessment of the technical feasibility of the system, design reviews for performance, early component-level and subsystem-level performance testing, system performance testing (performance measurement while running the system in test mode), and sizing and capacity planning.

This initial assessment will not attempt to pinpoint what specific kinds of services we need (e.g., predictive modeling, design reviews, performance testing, or all three), but will determine if we need to initiate a more substantive performance improvement project.

### The Purpose and Nature of the IIA

The IIA determines whether we need a performance improvement project in response to a particular system development project or a change, and provides a sense of the objectives, scope, issues, priority, timing and contact persons for that project. The IIA is a routine step which is performed early in each system project or when we anticipate a non-trivial change to a system or to its operating environment. Changes can occur which impact performance, without any system project being approved and initiated, so impact assessments are not driven only by the officially designated system development and maintenance projects. Performance projects can be expensive and time-consuming, and draw on scarce resources such as performance testers, load tools and performance testing labs, so this initial assessment provides an early filter which is important to determining where best to deploy these resources.

## *Appendix C: The Initial Impact Assessment*

The scope of the IIA is broader than performance testing and can include any performance-related services. This initial assessment does not necessarily attempt to pinpoint what kinds of services we need (e.g., predictive modeling, design for performance reviews, performance testing, or all three), but will determine whether a more in-depth performance improvement project should be initiated.

### Types of Impact Assessment

The impact assessment can take two forms:

(1) The situation where there is an existing infrastructure. Here the purpose of the IIA is to make an initial determination of the likely impact on this infrastructure, and whether the risks justify performance testing. For example, let's say that we are introducing a new application system into a complex environment. The impact assessment is important to both the success of the new application and the stability of the infrastructure, including its ability to support adequately the other applications using the same infrastructure.

The situation where there is no existing infrastructure. Here the purpose of the IIA is to determine what degree we need performance testing, if any, and to support the capacity planning for the new infrastructure for the system, determine if the system performance is adequate, and guide the system tuning to fit the environment.

### The Scope of the IIA

Since many activities can help ensure they meet system performance goals, the activities considered in the IIA are broader than just performance measurement. They include:

o Predictive modeling of the system performance.

o Assessment or re-assessment of the technical feasibility of the system, or the marketing feasibility for a commercial product. (For example, is it realistic for the product to beat the competition both on price -- which depends on the resources used -- and on performance?)

o Design reviews for performance, where we examine the behavior and performance characteristics of the system in walkthroughs to help identify likely bottlenecks.

o Design reviews for testability, leading to recommendations of features and hooks to build into a system to aid its performance measurement.

o Early component-level and subsystem-level performance testing, prior to the testing of the fully integrated system, which occurs late in most projects, after the design is locked in, and giving little time to react to surprises.

o Decisions on whether to measure the situation prior to introducing a change, in order to later develop a before-and-after comparison. This "before" picture can include the number of users, size of network traffic, application and database work load demands, current levels of resource utilization and the current system performance.

o System performance testing (performance measurement while running the system in test mode).

o Early identification of the tools, facilities and skills which we need to acquire or build, in order to prepare the test environment for the anticipated measurements.

o Sizing and capacity planning.

**Prioritizing the Performance Test Needs**

Despite their best intentions, most performance test teams can't do everything. The number of system projects and events where we could test performance is often overwhelming. The test team does not have all the equipment and tools needed for this testing even if we do have the time. In those unusual situations where there are no resource limitations, performance testers still need to focus their efforts in order to be effective. Without the right focus, the flow of events could distract the testers into numerous performance test efforts which with hindsight contribute little to the organization.

Which situations logically should receive the highest priorities for performance testing? The most likely candidates fall into these four categories:

(1) Resource-intensive systems (or subsystems, features, transactions or work demands).

o Systems in this category include one which place an intense demand on input/output facilities, are computationally intensive (processor bound), use unusually large amounts of memory or hard disk storage capacity, transfer large files or place a

heavy load on network bandwidth.

- For example, applications like videoconferencing and geographical information systems (GIS) have a name for being resource hogs – they typically place heavy demands on network bandwidth, and do enough file compression / decompression and perhaps also encryption / decoding to be computationally intensive too.

- As another example, consider an infrastructure which contains a mix of mainframes, database servers, personal computers, etc. We know that the existing work load places little stress on the mainframes, which have lots of spare capacity, but that the load heavily utilizes the database servers. A new system which uses the mainframes but not the database servers is unlikely to be resource intensive in this environment, and so would receive a low priority. On the other hand, if we re-designed this same application to move it off the mainframes and on to the database servers, it could shoot to the head of the line for performance testing.

(2) Timing-critical systems (or subsystems, features, etc.).

- These include real-time system where it must meet hard timing deadlines in life-dependent situations, such as an aircraft's flight control system.

- A more down-to-earth example is an executive information system (EIS), where queries from people like the CEO and CFO need speedy responses. Imagine the CEO taps in a query during a board meeting to answer a board member's impatient demand. The system administrator who cares about his job longevity does not want this query to sit in a queue on the network somewhere, stuck behind a huge print job.

- The category can also include highly visible systems or user groups, such as the home page of a Web site where many visitors download time is considered critical to success.

(3) Frequently used systems (or subsystems, features, etc.).

- This category high-volume network transactions, and the frequently visited pages in a Web site.

- For example, e-mail is ubiquitous. Each individual message may not be resource intensive in itself, and the timing is not super critical, but a swarm of messages can bring a network or a server to a crawl.

- As another example, I saw a situation where a new hypertext link was added to an existing Web page. The change seemed innocuous enough, but it brought the hosting system to its knees. It turns out that the new link was highly popular – everybody was clicking on it.

(4) Urgent changes or events. In the ideal world, project deadline dates are not a factor in prioritization. We would strictly base the amount of attention provided in performance testing on the first three factors above (resource-intensive, timing-critical, high-frequency utilization). Sometimes, though, a project which scores lower on these three factors takes priority, because of the urgency of its deadlines.

**What Situations Need to be Assessed?**

Prioritization is based on risk – where is the biggest risk of performance issues? Performance risks include response times which are too slow, inability to handle the number of users (throughput), inability to handle heavy loads, and inefficient use of expensive or scarce resources. Depending on how broadly we define the term "performance issues", related risks also can include inflexibility of the system to scale up (or down), difficulty in determining and allocating the optimal resources to run the system, difficulty in tuning and managing the system in operation as the work load mix changes, and an inability to pinpoint bottlenecks.

Not just new system projects introduce these risks -- any change can impact the situation. This means that every known change which is likely to impact performance needs to undergo a risk assessment and prioritization. Since there is a large volume of changes, this initial assessment and prioritization has to be brief and simple. Changes which may carry performance risks include the introduction of a new system or product, modification to an existing system, modification to the infrastructure used to support the system, such as the upgrade to a new version of an operating system, or a change in the user community or in the mix of demands that the users place on the system.

This means that the performance testers need access to information about a wide variety of changes – new product feasibility studies, change requests for existing systems, planned upgrades in the technical environment, and likely changes in the user community. We cannot plan performance testing for something we do not know about.

**Prioritizing Within Systems**

We assess the risk and prioritize not only entire systems, but also within them to help

focus the testing effort for a particular system. For example, consider a new system which uses a mix of mainframes, database servers, Web servers, local client personal computers and wireless technology, where the performance risk is high with wireless but low with mainframes. The wireless subsystem will have high priority, whereas the mainframe portion of the system may be exempted from performance testing.

In another situation, we might discriminate and prioritize based on the type of user. For example, consider a system with two distinct user communities: senior executives and lowly clerks. Rightly or wrongly, we are very concerned about the quality of service for the executives, but not for the clerks. If the system is slow, the clerks simply have to wait. We would assign a high priority to measuring the executive's likely performance. (Of course, to obtain a realistic sense of the executive's performance, we would have to ensure that the background noise during testing is representative of reality. This background noise includes the on-going hum of daily clerical activity, so we could measure the low-priority clerical response times fairly easily as a by-product of measuring the executives' performance.)

In a third situation, we could prioritize the test efforts within a system based on the events or transactions the system receives. It is not unusual for the most popular 10% of all the types of transactions to account for 90% of the transaction volume, or the most resource-intensive 10% of events to gobble up 90% of the system resources. Unless we need unusually precise measurements of system performance, we can safely ignore the 90% of all transactions which consume only 10% of the resources.

**The Prioritization Process**

Since change is way of life, we need to periodically re-assess the overall situation and re–visit the priorities. This re-assessment is usually done monthly, and may be triggered more frequently by new information. The performance test team can coordinate the prioritization activity, if there is a separate team. It is done in conjunction with the system owners or custodians, such as development project leaders or business unit managers, and in conjunction with the resource suppliers, such as network administrators or data center capacity planners.

In the prioritization process, the four risk factors which were mentioned earlier (resource-intensive, timing-critical, high frequency, and deadline urgency) are each subjectively ranked on a scale of high, moderate or low. Quadruple-threat activities are the ones which score high on all four factors, and these receive the highest priority and the most attention. Minimal-threat activities score low on all four factors, and these will receive no attention. If a project or event scores one high evaluation among the first

three factors (resource-intensive, timing-critical, and high frequency), then it should receive some attention from the performance testers. At minimum, we can track the project or event to ascertain whether and when it turns into a multi-threat activity.

**When to Conduct the IIA**

When should we undertake an impact assessment and prioritization as described above, and when can we kick off the performance testing project if it is justified? The answer is that we want to select projects and events (such as non-trivial changes in work load) for performance testing as early as we can. For a new system project, we usually have enough information to assess its likely performance impact by the end of the system requirements definition, and definitely by the time the high-level system design is done. For an anticipated change to an existing system (such as the expansion of the number of system users), ideally the change request will provide enough information to be able to prioritize. As system projects progress and more becomes known about them, we typically want to fine -tune the performance testing objectives and strategy. This is part of the periodic re-assessment mentioned earlier.

**Potential Funding Issues**

Perhaps the budget should not cloud the technical issues, but resource constraints are ever present, and the availability of funding can be an issue in initiating performance testing projects. For example, is the test effort going be funded from a centralized budget which the performance test team is awarded, or funded out of the system development budget, or underwritten by the user community?

Since the source of funding usually influences who gets to call the shots, it can have an impact on the project objectives. If the performance team is funded from the individual budget which is “owned” by each project, the analysis tends to be project-specific. (“What’s the response time for this feature in our system?”) If instead the team is funded from a central budget, the analysis tends to be more enterprise-wide. (“What’s the impact of this feature on the existing infrastructure and on the users of other co-resident systems?”)

# *Appendix D: Roles and Responsibilities*

## APPENDIX D. ROLES AND RESPONSIBILITIES

### Overview

This appendix discusses who's involved in performance testing, and their perspectives and motivations. In a typical enterprise, which is not a vendor or supplier but is primarily a user of technical products and services provided by others, the people involved in performance management include (a) the senior managers, (b) the people who run the operations such as system administrators, (c) the people who monitor the operations such as auditors and capacity planners, (d) the system architects, designers and developers, (e) the testers and (f) the users.

### The Role of the CIO and Senior Managers

The business managers are interested in providing operational continuity, improving their competitive position and capitalizing on new opportunities in areas like e-commerce. Most senior managers willingly invest in new technology for promising opportunities and want the information to make smart decisions. Companies can waste resources if they cannot evaluate alternatives correctly, but measuring returns on e-commerce projects can be daunting. Predicting customer behavior is difficult, because using the Web to do business is still new. Benefits and costs are difficult to measure, and the rapid pace of change does not help, as it means there is little comparable prior experience. In addition, the right choices of metrics are different for each company, because the strategies, structures, and systems are different. Understanding the business goals comes first, since this understanding drives the choices of metrics.

Understanding the motivations of these managers improves our chances of winning their support. Which of these motivate your CIO and senior managers, and which are their highest priorities?

- Improve ROI.
- Reduce operating costs without imperiling service.
- Save money and right-sizing networks and infrastructure.
- Buy what you need, by evaluating vendors' claims before you commit.
- Improve user satisfaction.
- Win by providing better service to the business functions.
- Quicken delivery and improve responsiveness to users' requests.
- Reduce risk.

## *Appendix D: Roles and Responsibilities*

- Gain confidence that security controls work and are adequate.
- Know the impact of changes before they are introduced.
- Check interoperability and that the infrastructure is seamlessly integrated.
- Understand better where the stress points, weaknesses and vulnerabilities are located.
- Ensure the disaster recovery plans are “not a disaster”.
- Provide leadership.
- Lead the organization to capitalize on technology-based opportunities.
- Develop policies and goals for performance and reliability.
- Negotiate and monitor compliance with SLAs.

**The Role of System & Network Administrators**

Which of these motivate your system and network administrators?

- Maintain operational integrity.
  - Keep everything running smoothly – applications, servers, firewalls, e-mail, printers, phones, networks, etc.
  - Satisfy the users: meet the service level agreements (SLAs).
  - Ensure resource utilization is cost-effective.
  - Comply with policies, standards and operating procedures.

- Manage change.
  - Implement application changes safely and responsively.
  - Check infrastructure upgrades and configuration changes in the safety of the test lab, before deployment.

- Improve the levels of service provided.
  - Tune the infrastructure to improve performance and reliability.
  - Increase security.
  - Improve resource utilization.
  - Provide advice to users and other technical specialists – e.g., DBAs.
  - Tune systems, databases and networks.
  - Keep up to date with technologies, methods and tools.
  - Troubleshoot and resolve problems.
  - Manage vendor relationships.
  - Monitor and report the quality of service that is provided to users.
  - Perform capacity planning.

# *Appendix D: Roles and Responsibilities*

## The Role of the Testers

- Validate that systems and changes work as planned.
- Set up large-scale tests: simulate the real world in the test lab.
- Provide better test coverage.
- Re-test quickly and cheaply after changes.
- Find discrepancies (bugs).
- Test at and beyond the breaking point.
- Improve the ratio of bugs found in testing vs. live operation.
- Provide more complete and timely information to decision makers about the state of a system, for example, about its readiness for release to live operation.

## The Performance Test Team

Some reasons why performance testing is often not done well is the testers' lack of in-depth expertise in the factors that affect performance, the testers' lack of suitable equipment for performance testing, and the testers' demanding work load to get other types of testing completed.

For all of these reasons, an interdisciplinary team should tackle the performance test planning composed of experts in data center operations, vendor equipment technical specialists, network administrators, together with the testers. Other specialists such as application developers and database administrators will need to be available, as consultants to the performance test team.

The experts need to share information and cross-educate each other in order to identify the performance test goals, issues and approach. They will be involved in designing cross-platform performance test cases. The experts can also identify the equipment needed for performance testing and facilitate this equipment being made available for the testing.

## A Caution

These people often are poor performance testers: performance engineers and capacity planners, architects and developers, project managers and system developers, and operations specialists like system administrators and network engineers. Reasons: bias and over-emphasis on topics within their comfort zones; lack of understanding and under-valuing of the testers' role; failure to know what they don't know; inability to see from an enterprise perspective vs. a narrow one; inability to escape local politics.

# *Appendix E: Performance and Robustness Testing Methods*

## APPENDIX E. PERFORMANCE AND ROBUSTNESS TESTING METHODS

### Approaches to Testing

There are many variations of testing within the broad framework of performance testing. Knowing these variations is important, because selecting the right ones for a particular situation is essential for developing an effective test strategy. This section names and explains the main variations. There is some overlap in the following types of testing.

Many projects have both performance and robustness testing goals, and sometimes it is difficult (and pointless) to differentiate between these two types of goals, especially in the area of stress testing. So in the following list, I have not tried to distinguish the performance testing methods from the robustness testing ones; I have mingled them together. There is some overlap among the various types of testing on this list, and we may employ several of them within one test project.

In addition, there is no universal and consistent set of terminology, and many organizations have their own terms such as “work load testing” and “sweet spot testing” (both of which are terms used inside divisions of Hewlett-Packard). The way I am using terms in this book is the mainstream, the common vernacular, to the extent that there is a common term being used.

The main types of performance testing and measurement are:

**1.0 Testing which is driven by what we want to measure.**

- 1.1 Response time testing
- 1.2 Throughput testing
- 1.3 Availability testing
- 1.4 Measurement of resource utilization
- 1.5 Capacity testing
- 1.6 Measurement of delays (latency)
- 1.7 Measurement of losses in networks
- 1.8 Error rate measurement

**2.0 Testing which is based on the source or type of the load.**

- 2.1 Usage-based testing
- 2.2 Standard benchmark testing
- 2.3 Load variation testing

2.4 Ramp-up testing
2.5 Component-specific testing
2.6 Calibration testing

**3.0 Testing which seeks to stress the system or find its limits.**

3.1 Scalability testing
3.2 Bottleneck identification and problem isolation testing
3.3 Duration or endurance testing
3.4 Hot spot testing
3.5 Spike and bounce testing
3.6 Breakpoint testing
3.7 Rendezvous testing
3.8 Feature interaction / interference testing
3.9 Deadlock testing
3.10 Degraded mode of operation testing
3.11 Synchronization testing
3.12 User scenario, bad day and soap opera testing
3.13 Disaster recovery testing
3.14 Risk-based testing
3.15 Hazard or threat identification
3.16 Environmental testing
3.17 Compatibility and configuration testing

**4.0 Testing which focuses on the impact of changes.**

4.1 System change impact assessment
4.2 Infrastructure impact assessment
4.3 Baseline testing
4.4 Volume testing
4.5 Parallel testing
4.6 Live patch and change testing
4.7 Extreme configuration testing

# *Appendix F: Challenges in Performance and Robustness Testing*

**APPENDIX F: CHALLENGES IN PERFORMANCE AND ROBUSTNESS TESTING**

## Why is Performance Testing Difficult?

Performance testing is dead simple. All it requires is the right mix of demands (work load) to use in testing, the right test environment, the right tools and techniques, on-target insights into what to look for and where to look; an adroitness in handling deadline pressures, the ability to work with sometimes uncooperative architects, developers, system administrators and vendors, the ability to live with a great deal of unknowns and uncertainties, the right interpretation of the measurements, a way of translating results from the test lab into valid predictions about performance in the live operating environment, a knack for pinpointing and isolating bottlenecks, and inspiration and good luck in resolving the performance problems which the testing uncovers. Good luck.

Frankly, performance testing can be a real pain. It often is not done well, and more often it is not done at all. With more frequency than many testers would like to admit, the measured performance in the test lab bears virtually no discernable relationship to the actual system performance in live operation.

## Common Issues of the Testers

- Frequently, the performance goals are vague.
- Managers often lack awareness and understanding of what performance and robustness testers do – have to re-justify.
- Poor cooperation from developers and system administrators in resolving problems.
- Performance, robustness and reliability are often afterthoughts in design and in testing.
- Often must persevere with tool limitations and under-funded test labs.
- Expected to catch all the significant bugs, a sobering responsibility.
- Typically labor under deadline pressures.

(This is not a complete list.)

## *Appendix F: Challenges in Performance and Robustness Testing*

There are several difficulties listed below which can complicate a performance test:

### A. Background Knowledge

*A1. Understanding of the System and its Context*

To competently assess whether a system's performance and robustness is acceptable, we need to understand its behavior. The desired knowledge base includes several areas which no one person may know well – (a) the system's functions, (b) its internal architecture, (c) how it is going to be used, (d) the technical environment or support infrastructure in which it will operate, (e) the perceived risks and vulnerabilities, and (f) the managements' or clients' expectations for performance and robustness.

*A2. Slope of the Learning Curve*

It is hard to test effectively without sufficient time to prepare. Although the performance and robustness testing may not happen until late in a project, it is important to involve the test team early so that they have time to climb the learning curve and thoroughly understand the system, its context and issues. The testers have enough time to select and learn the right tools, and develop automated test cases – a notoriously lengthy process, and acquire the most suitable test equipment. If they become involved early, the testers also can play an important preventive role in avoiding later surprises. (See the section entitled: "Avoiding Surprises".)

*A3. Availability of a System Model*

Unless a system is extremely simple, we cannot evaluate its performance and robustness without a model or at least a mental map of how it works internally. In other words, we generally cannot test only from a black-box perspective but need to "look under the hood". For example, complex systems usually have overwhelming number of factors which could be pertinent to system performance and robustness, and which we could monitor Without a sense of what's important – what to look for, we will collect large volumes of meaningless numbers.

If a cohesive, comprehensible model already has been developed by the system designers, then we can utilize this model. The problem often is the adequacy of the models available – it is fair to say that many designers of complex systems do not understand the intricacies of how their systems actually work.

*A4. The System Scope and Boundaries*

## *Appendix F: Challenges in Performance and Robustness Testing*

We have to determine exactly what "it" is (i.e., the system we are testing). If a manager gives us the directive: "Test the system's performance", we may have to respond by asking: "Just what do we mean by the term 'the system'?" At first hearing, this point may sound dense, which it's not.

Sometimes there is a vagueness about what a testable version of a system includes. If the system is part of a greater integrated super-system, can we somehow isolate that system for testing, or does it make sense only to test the whole super-system?

A system can exist in several versions concurrently. Within one version, different subsets of the features will be enabled at different points in time, while other features are unavailable at that time. The system might be tuned or optimized for this particular test, or deliberately not.

Many systems are designed to incorporate an indefinite number of third-party plug-ins, some of which have not been invented as yet. Third-party software plug-ins include network drivers, middleware and many other types of software. These drivers may not even be needed by the "system" we are testing, but instead are used by other applications which normally run in the background. We'd need to determine which third-party drivers to enable and use during testing, which to enable but not use, and which to not enable.

Another issue is whether the system should be tuned and optimized for the particular test load and environment used in testing. The test results may be mis-leading (too good), and the system sub-optimized for other different loads and environments.

### B. The Testware

*B1. Adequacy of the Test Facilities*

One issue is what facilities we can use to run the performance and robustness test. Ideally, the environment for testing will fully mirror the live operational environment, but this can be extremely expensive or infeasible. Or we could perform the performance test in the live environment -- which risks slowing all the other users to a crawl, or worse, interfering with and corrupting their work.

*B2. Adequacy of the Test Work Loads*

Another issue is how to select or develop the work loads, the mixes of demands to be placed on the system during performance measurement.

*B3. The Overhead and Logistics of Testing the System*

## *Appendix F: Challenges in Performance and Robustness Testing*

Sometimes the day-by-day mechanics of testing are all-consuming. We are absorbed by setting up the test environment, developing loads, writing and debugging automated test cases, running ad nursing along the test, etc. We have no time or energy to think, strategize, question what we are doing and why, or consider how to improve our testing effectiveness.

An example is the issue of how to place the load on the system which we are testing. Should the test team use an automated load test tool? (These tools are expensive and can be tricky to use.) Or can the test team instead round up a hundred volunteers to bang away on the keyboards, in order to stress the system while we measure its performance? The "hundred volunteers" approach carries costs of its own.

Another example: should we run a duration test for four hours? Or eight hours? 24? 72? If we are busy with the mechanics of just getting the test to run, we don't have time to think through the trade-offs among these alternatives.

*B4. Unanticipated Glitches in the Test Environment*

The occurrence of glitches is high in many test labs, especially when different equipment is patched together or the testers are not very familiar with the equipment.

The glitches include mis-connected equipment, software incompatibilities, operator errors and even a moth stuck between two electrical contacts. (This famous moth, which is now on display in the Smithsonian museum, originated the use of the term "bug" for a hardware or software defect.)

### C. The Live and Test Environments

*C1. Environment-Specific Issues*

Every system exists within an ecosystem, and cannot function or have much meaning independently. Performance and robustness testing are very environment-specific, perhaps more so than other types of testing. These types of testing have a very different flavor on an IBM batch mainframe than in an Internet service provider's packet-switched network or for a video game.

There often are environmental complications in performance and especially in load and stress testing, and resolving these complications requires considerable expertise in the particular technical domain (e.g., Unix system administration). Planning, configuring, running the performance test and interpreting the results often requires the involvement of people who are highly knowledgeable about the specific environment we are testing.

These people tend to be scarce and busy.

*C2. Mixed Environments*

Environment-specific skills are not easily interchangeable. A person who is highly knowledgeable about Unix may know little about Macintosh, and vice versa. A database administrator does not know how to run a network except in her home office, and a network engineer can't manage the corporate data warehouse.

However, most organization's operational environments mix multiple vendors, multiple technologies, multiple operating systems, multiple databases, multiple network protocols, etc.

This means the testers may be dependent on several different technical specialists from different organizations, and the effort to coordinate their activities in a complex environment can be considerable.

*C3. The Impact of New and Improved Technologies*

The rapid rate of innovation in software, hardware, networks and databases means an on-going stream of opportunities to improve system performance and robustness, and frequent upgrades to systems and their environments.

The NGT (next great thing) brings enthusiasm and sometimes a misplaced euphoria. The new-fangled whatsit will run so fast that all our performance problems will disappear, and in our confidence we believe there's no need for performance testing. Hindsight tells us that the NGT is rarely a panacea, but in the enthusiasm of the moment it is easy to overlook this.

We often incorporate new technologies, such as wireless and security devices based on human physiology, into existing infrastructures, where they can interact in unforeseen ways with what's already there.

The continuous stream of opportunities to upgrade means that environments are not static and may not stay stable for very long. Every "opportunity" has performance and robustness implications and thus triggers a never-ending demand for re-testing performance, robustness and stability.

*C4. Large Numbers of Testable Configurations*

Today, many application systems are highly configurable. So are their support software, hardware platforms, databases and networks.

## *Appendix F: Challenges in Performance and Robustness Testing*

When we consider all the different possible combinations of the internal switch settings in an application, all of which potentially can effect the system's performance and robustness, and all the different combinations of hardware platforms, network topologies and support software, the number of ways the system can be used is astronomical.

The system and environment configuration settings may not be easy to observe from the outside, and the configuration may change during the flow of work, as events change the switch settings, without the testers being aware of it.

### D. The Testing Tools

*D.1 Expense and Complexity of the Tools*

The tool acquisition cost, if a tester uses a commercially available tool, is only the tip of the iceberg. The total cost of ownership (TCO) is much higher, sometimes by a factor of a hundred. The overhead to develop and maintain the automated test cases can be horrendous.

Many testing tools are rich in capability but hard to master and utilize well.

The tools often complicate the performance and especially the load and stress testing. Automated tools can time-out because response times during testing exceed the defaults to which the tools have been set (see the discussion of load testing tools later in this book). Or the load test may fail because of the heavy load and crash the test platform, wiping out the captured measurements and suspending the test until it is manually re-started.

Many organizations must build their own testing tools because nothing is commercially available which will meet their needs, and the expense can become extremely high. Most large vendors build their own tools, and it is not uncommon in firms like Cisco to find full-time teams of dozens of test automation specialists who build and maintain tools, but do not test.

*D.2 Tool Limitations*

Performance, load and stress test projects are very tool-dependent: most of the work cannot be done manually. In many projects, though, there are many things that the available tools will not do or not do well, leading to compromises and a need to make assumptions. And the tool users may not know enough about a tool to realize what these compromises and assumptions are.

Many of the tools are quirky (buggy), not easy to use and not particularly well supported by the tool vendors, despite the high fees they often charge.

*D.3 Mis-Use of Tools*

Testers can unknowingly mis-use load testing tools to apply loads in ways which are not representative of reality. These test bugs happen much more often than many testers realize. See, for example, the later discussions in the sections entitled: “The Metronome Effect” and “Limitations and Common Mis-Uses of Tools”.

## E. The Test Methods

*E.1 Testability*

Testers are often frustrated by their inability to observe the behavior of a system. There may be an internal hidden switch or counter whose value is transient – here one millisecond and gone the next – but with no way for the testers to access this switch before its value changes.

Most systems are designed and built with little attention to their testability. See the later discussion in the section entitled: “Designing for Testability”.

*E.2 Deciding What to Measure*

We may have a strong model and a good understanding of a system and its environment, but still not know what to look for, especially before we have had much experience with the system.

Consider what it takes to measure the response time for a Web page to download. At first hearing, it sounds simple – one test case can do the job. Then we start to consider all the factors which might influence response time (browser, connection speed, time of day, number of other concurrent visitors to the Web site, etc.) The number of measurements needed, in order to obtain a realistic sense of the delays experienced by users, increases dramatically.

Let’s now say that we are not just interested in the response, but in the relationship between response time and the amount of memory allocated to process download requests on the Web server. The number of influencing factors to consider increases again.

*E.3 Lack of Repeatability*

## *Appendix F: Challenges in Performance and Robustness Testing*

The fundamental laws of physics seem to be suspended in performance measurement. If we re-run a performance test case under exactly the same conditions as before, we expect to get exactly the same performance measurements. To our surprise, though, the same behavior does not happen again. Factors have changed which we cannot see or control, but which affect the system's performance.

This means we can't simply measure something once and be done with it, but we may have to take a sample of measurements (which can be sizeable). Then we need to decide which value to use: the simple average of all data points, the average after cleansing the collected data points of dubious values, the mode or the median, etc.

*E.4 Interpretation of Results*

Sometimes it is not clear how the performance data which the performance testing team captures will actually be useful. The data which we do capture may not be sufficient to develop clear and unambiguous answers to the managers' real-world business questions, such as: "How will this system improve the productivity of my business unit?"

*E.5 Problem Isolation*

When we test an application, we cannot run it stand-alone. It needs to be integrated and tested with support software such as an operating system (OS), not to mention the supporting hardware, networks and databases. The question is whether we can isolate the performance of the application from that of OS, DBMS or other support software, or from the unique interaction of that particular OS, DBMS and application. Another question is whether we even want to try to isolate the problem. And if so, what level of effort is justified – the trade-off of testers trying to pinpoint a problem versus turning it over to experts.

*E.6 Validity and Credibility of the Results*

Many testers' predictions of performance and robustness do not match the users' actual experience in live operation. Often, this mis-match is caused an accumulation of myriad little inaccuracies between the test lab and the real world. The testers are unaware of many of the inaccuracies. Where the difference is known, it may be a deliberate compromise of reality which is needed to make the testing feasible. The process of adjusting the lab findings to take into account the myriad differences is not well understood.

### F. Project Management

*F.1 Late Occurrence in Projects*

## *Appendix F: Challenges in Performance and Robustness Testing*

In addition, performance testing typically happens towards the end of projects, only after we have tested and fixed the basic system functionality. As such, we conduct the performance testing under stress conditions (no pun intended), when the test team is overworked and faces deadline pressures to get the system delivered.

*F.2 Testing without Pre-defined Exit Criteria*

Performance measurement is often open-ended, not a one-time activity. There are an indefinite number of iterations of tuning the system in response to the reported performance, then re-measuring to see what the cause-and-effect impact of the tuning is on the system performance. Budget and schedule overruns are common on performance testing projects.

*F.3 "Trivial" Changes which have Widespread Consequences*

Seemingly tiny and unnoticed changes can have a major impact on system performance and reliability. For example, I saw a situation where a Web site administrator added a new link to his Web site. The new link allowed the user community to access a popular external Web site containing trendy gossip and jokes. A trickle of use of the new link quickly became a flood and the entire Intranet of the organization slowed. (The link was hastily removed.) The flow of these changes is never-ending.

*F.4 Specialized Expertise Required*

Performance and robustness testing can demand specialized knowledge of the network, database, operating system, hardware and performance testing tools, which the feature testers do not have. This is not a complete list: we could mention more complications of performance testing too.

*F.5 Coordination of Specialists*

Because of the narrow, deep levels of technical expertise available, performance testers can find themselves playing the role of general contractor in house construction. The testers provide a de facto center, an information clearing house, and they coordinate the tuning and debugging activities of different topical experts. The disparate specialists often have different perspectives, priorities and biases. Working with them can be like herding cats.

Without coordination, it is easy to sub-optimize. The Unix specialists will naturally tune and optimize the parts of a mixed Unix-Windows environment which they can see and

understand, and the Windows system administrators (to whom Unix is a mystery) will do the same thing for the Windows portions, but they may work at cross purposes. In house construction, everyone has heard stories like the one where an electrician unknowingly drills a hole in a wall – and through a hidden pipe which a plumber just installed. Sometimes this coordination role is not one the testers anticipated, so no overhead has been factored into the schedule and budget to get it done.

*F.6 Lack of Candor*

Management must encourage lower level team members to speak out, be critical.

*F.7 Unexamined Information*

Testers must not accept information uncritically and probe beyond the information provided.

The people who present the architecture must be those familiar with it, the people who present the operational profiles must familiar with how the users really work, and so on..

*F.8 Lack of Budget and Time*

Realistic budget and time needs to be allocated in the early planning stages.

*F.9 Lack of Buy-In*

The organization needs to understand the importance of testing performance.

On the other hand, expectations need to be realistic.

# *Appendix G: The Test Automation Framework*

## APPENDIX G. THE TEST AUTOMATION FRAMEWORK

### What is an Automation Framework?

The automation framework is the blueprint for the coordinated set of test tools, equipment, procedures and support people needed to make test automation effective and efficient. In the words of Bob Poston: "Newcomers to automated software testing often think that all the automation they need is built into a capture-replay tool that runs test cases. As people move further into automated software testing, however, they see that test case execution takes care of only one small part of their work. Test cases must be designed and created before they can be run and results captured. Additionally, test cases should be evaluated after they are run, so the tester and the developer can confirm that the software functioned or failed to function as expected."

Many automated test groups do not consciously design and manage the framework, but by default allow a disparate collection of testing tools and processes to evolve over time. The result, like any informally designed and patched-together system, is a hodge-podge of partially effective tools which are not particularly well integrated together. In addition, the automation of test case creation and evaluation are not widely understood and practiced.

Today, many tool vendors provide useful solutions for individual parts of the framework, but no one tool vendor has a good overall, integrated solution, despite the vendors' claims, except for the simplest and most straightforward test situations. This means it is up to the test organization to develop their own framework, and the select and fit vendors' products into the framework, or alternatively build their own, or most likely have a mix of vendor-provided and home-built tools.

### Why do I need an Automation Framework?

With a framework:

- The various test facilities can be smoothly coordinated to work together.
  - The test tools can be coordinated to work with the test facilities.
  - The test tools, test support tools and test support processes can be coordinated.

- People understand their responsibilities for automated testing and automation support.

# *Appendix G: The Test Automation Framework*

Without an appropriate framework:

- o The integration of the automated testing with the testing support tools tends to be cumbersome and inefficient.

- o The testing efforts tend to be disorganized, with a jumbled heap of inconsistent and redundant test cases and processes.

- o Everyone creates his or her own way of automating their testing efforts.

All too often, a test automation project is organized like the worst programming project you ever saw. A few busy test professionals are given a copy of an automated testing tool and, with no training, told to "go do it" (write some automated test cases). The result inevitably is a disorganized, uncoordinated jumble of unmaintainable automated test cases.

While organizing the automation framework can sound like bureaucracy, overhead and delay in an automated testing effort, it is an important first step in getting organized for automation.

**Components of the Framework**

The framework for automation includes these six components:

*(1) The System(s) to be Tested.*

*(2) The Test Equipment and Facilities.*
- Diagnostic Utilities.
- Configuration Management.
- Processes for Loading Systems to be Tested.

*(3) The Test Case Library*
- Data Base Organization of this Library.
- Guidelines for the Use of the Library.
- Maintenance Procedures for the Library.

*(4) Automated Testing Practices and Procedures.*
- How to Use the Test Tools and Test Case Libraries.
- How to Write and Maintain Automated Test Cases, etc.

*(5) The Tools.*
- Automated Testing Tools.
- Support Tools.
- Integration of Automated Testing with the Support Tools (e.g., requirements tracing, defect tracking, management reporting).

*(6) The Peopleware* (Support Roles for Automation).

This section provides brief descriptions of the tools shown in the boxes on the preceding diagrams. Most of these tools are described in more depth later, along with examples of commercially available tools.

# *Appendix G: The Test Automation Framework*

## A. Overview of the Automation Framework

This diagram shows the overall context of automated testing.

<<See the separate Test Automation Framework handout for diagram pages to insert here.>>

## *Appendix G: The Test Automation Framework*

### B. Front-End Test Tools

<<See separate handout for diagram pages to insert here.>>

## *Appendix G: The Test Automation Framework*

### C. Test Environment Management Tools

<<See separate handout for diagram pages to insert here.>>

## *Appendix G: The Test Automation Framework*

### D. Back-End Test Tools

<<See separate handout for diagram pages to insert here.>>

## *Appendix G: The Test Automation Framework*

### The Role of the Framework Builders

The builders of the automation framework should spend much of their time up front: listening to the clients (i.e., the testers who will using the automated test facilities), understanding their needs and how they test systems, scrutinizing the feasibility of automation alternatives, forming a practical vision of the automation framework, and creating a blueprint.

The framework builders are usually also responsible for implementing the framework, not just designing it, i.e., by building or installing facilities according to the blueprint for automated testing. The construction phase of the framework development can include in-depth tool evaluations, integrating tools together from different sources, converting test data bases, etc.

The effort involved in building an effective framework should not be underestimated. Typically, the builders will find themselves involved in many activities which require both time and skill. The framework builders:

- Focus on the requirements of those who will use the automated testing facilities.
- Need to understand the types of systems being tested, the scope of the test automation, the user-testers' main testing issues, requirements, and expectations. The architects also study the context of the automation - the entire enterprise of which the test organization is a part.
- Must thoroughly understand and document the areas (domains) for which the framework will be built and to learn the testers' requirements in detail.
- Advocate an organized test environment. (This is an important part of the framework builder's role).
- Need to bring an extensive knowledge of the fielding of testing, to be able to design the right framework and select intelligently from an unlimited spectrum of choices.
- Outline the desired behaviors and capabilities of the overall framework.
- Prepare architectural-level designs depicting the domain characteristics and technology structure.

- Expand the detail and refine the framework model to converge on the final design.
- Consider the look and feel of the system - the most appropriate style for the user interface.
- Build prototypes for selected components of the test automation framework, if they are needed.
- Perform risk assessments on the framework.
- Develop guidelines and handbooks for users of the automated test environment.
- Specify the tools and methodologies, which are needed as components of the framework.
- Participate in testing and acceptance reviews of the automated test environment, to the extent the clients (the testers) desire.
- Assist the users (the testers) with the migration to the new automated testing environment.
- May be involved with the training of the users of the automated test environment, as needed.

**The Existing Infrastructure**

Automation cannot be done successfully in isolation. There is a broad and rich diversity of corporate cultures, software engineering practices, and existing test facilities. Test automation needs to be customized and carefully grafted on to the existing organization (assuming we want evolutionary change and not revolutionary change).

Since the automation framework is dependent on the culture of the organization, it needs to encompass (or deliberately exclude) what already exists:

- Existing Test Plans and Test Cases.
- Existing Test & QA Processes.

- Existing Test Team Organization.
- Existing Human Resources and Skills.
- Existing Tools. (For test data management, problem reporting, version control, etc.)

**Integration of the Test Tools**

**Types of Tool Compatibility**

There are three main types of compatibility and integration which need to be considered when selecting or building a test tool:

(1) Compatibility with the technical platform, such as the operating system, data base management software, network software, etc.

(2) Compatibility with the development languages and tools. A particular automated testing tool may work just fine with applications developed in one language, let's say Visual Basic, but cannot "see" objects like buttons in applications developed using other languages.

(3) Compatibility with the other interconnected tools which constitute the test environment, such as requirements management tools, test case repositories, problem reporting and tracking tools.

Incompatibilities among any of these areas can seriously impede the effectiveness of the test tools.

There may also need to be other areas of integration -- across different hardware platforms ad devices, operating systems, networks, etc., or across different existing and incompatible test case repositories and test data bases.

The fact that not just one but multiple types of integration is needed is usually major complication in building the test facilities.

**Development Tool Compatibility**

How do we determine what test tools work the best (or even at all) with a particular development tool? (Note that by asking this question we have considerably and

artificially simplified the multi-dimensional compatibility issue which was raised above, which included the technical environment, the development tool, the automated test execution tool, and the elated test support tools. Even the two-dimensional compatibility question of how well test and development tools work together is tough enough, however.

There are five ways to tackle this compatibility question:

- Vendor-Neutral Published Evaluations
- Test Tool Vendor Certifications
- Development Tool Vendor Certifications
- Shared Knowledge of Users
- Hands-On Pilot Projects

# *Appendix H: Test Automation Activities*

## APPENDIX H. TEST AUTOMATION ACTIVITIES

Since many factors need to be considered when you develop your performance test automation strategy, it is easy to become distracted. Focus is important, so you need to identify what is critical to your strategy.

I have listed over 100 activities below that testers and test managers typically undertake when automating their performance testing. This list was compiled by merging the major activities of twenty different test automation strategies. The scope of this list is broader than just performance test automation, as it is usually beneficial to integrate the performance test automation with the functional test automation. And automation often drives other ancillary activities such as revising test procedures. Though it is broad, the list is not all-encompassing: please add any critical activities you need which are not already on the list. There is some overlap among these activities, and they are not presented in any particular sequence. While some may be irrelevant in any given situation, many are likely to contribute to the success of your test automation and a few will be critical.

Review the following list and label each activity as either (a) critical (b) important, (c) minor or (d) irrelevant to you. You should not label more than 15 to 20 as critical.

### A. The Overall Approach

1. Developing your automation objectives and goals.
2. Prioritizing the automated test needs, across systems or across test projects.
3. Sketching a blueprint of the infrastructure: the test automation framework.
4. Recognizing and checking assumptions.
5. Developing the risk assessment methodology.

### B. Support

6. Identifying the major constituencies who have vested interests in test automation, and their success factors.
7. Selling the test automation strategy to senior management; obtaining management commitment.
8. Obtaining system developers' buy-in.
9. Obtaining system users' buy-in.
10. Obtaining system administrators' and operations management's buy-in.

### C. Review of the Current Situation

11. Using a test information gathering questionnaire to guide automation.
12. Conducting user satisfaction surveys regarding the test facilities.

13. Assessing readiness for test automation.
14. Taking an inventory of tools already installed.
15. Appraising current test automation skills.
16. Assessing the effectiveness of current or prior test automation attempts, and the lessons learned or to be learned from them.

### D. Justification of Test Automation

17. Performing a cost/benefit analysis for functional test automation.
18. Justifying the performance and robustness testing effort

### E. Assessment of Automation Effectiveness

19. Setting criteria to assess the effectiveness of automated functional testing.
20. Setting criteria to assess the effectiveness of automated performance and robustness testing.
21. Developing techniques for assessing test effectiveness.
22. Implementing the metrics needed to track the effectiveness of test automation.

### F. Outsourcing

23. Deciding whether to outsource functional testing.
24. Deciding whether to outsource performance and robustness testing.
25. Deciding whether to outsource other automated testing or support tasks.

### G. Automation Start-Up

26. Identifying the key automation start-up decisions and the process for making those decisions.
27. Choosing the pilot project .
28. Spreading propaganda and success stories to promote interest in test automation.
29. Identifying any automation start-up issues to be resolved before proceeding.

### H. Planning for Test Automation

30. Planning for the test resources.
31. Calculating the impacts of automation on the existing resources..
32. Estimating the automation work effort.
33. Preparing the automation project schedule.
34. Preparing the automation project budget.
35. Developing the automation work plan template including major milestones.

### I. Automated Functional and Performance Test Plans

36. Developing a standard test plan outline.

37. Developing a test methodology and project template.
38. Defining the format and documentation requirements for test projects.
39. Defining quality gateways.
40. Establishing test entry criteria.
41. Developing methods for testing the impact of changes to a system.
42. Developing regression testing guidelines.
43. Setting the test coverage targets and test completion criteria.
44. Determining the work scope: developing guidelines for what to test manually and what to automate.
45. Selecting the testing techniques appropriate for each situation.
46. Developing a test plan review checklist.

### J. People and Organization Factors

47. Organizing the teams for the test automation strategy, feasibility study or proof-of-concept pilot project.
48. Defining the automated testing roles & responsibilities.
49. Documenting formal job descriptions.
50. Establishing the role of the test librarian or test repository director.
51. Conducting staff orientations on how their work will change with test automation.
52. Training and skill upgrading in preparation for automation.
53. Hiring experienced consultants to guide the automation effort.
54. Deciding on centralized vs. decentralized automation.
55. Planning for a small, central advisory team to assist decentralized test automation projects in various parts of the organization.
56. Deciding who tests the testers, e.g., by defining auditability and establishing audit trails.

### K. Tool Utilization

57. Developing the test tool requirements.
58. Performing the test tool evaluation and selection .
59. Selecting the automated testing platform.
60. Making the tool build versus buy decision.
61. Making the commercially available versus open source tool decision.
62. Identifying the types of testing and test-related tools needed.
63. Deciding which already-installed tools to keep and which to replace.
64. Implementing a front-end interpreter tool, to allow testers without programming skills to use the automated testing tools.
65. Building simulators or prototypes for use in testing.
66. Using tools to facilitate walkthroughs and inspections.
67. Automating the consistency checking GUI windows and we page layouts, for

compliance with the look-and-feel usability guidelines.
68. Implementing network sniffing.
69. Implementing orthogonal array or all pairs tools to identify test configurations.
70. Implementing database integrity checking tools.
71. Using automated tools for checking security controls.
72. Determining how to integrate the test tools and test support tools.

### L. Test Policies and Procedures

73. Establishing automated testing standards.
74. Developing guidelines for what to automate and identifying the most fruitful areas for automation.
75. Developing policies for exploratory vs. structured testing.
76. Developing policies for early vs. late test automation on projects.
77. Developing a test glossary of terms.

### M. Automation of System Requirements and Design

78. Providing an automated requirements management capability.
79. Providing traceability from system requirements to test cases, from test cases to system versions, etc.
80. Designing for performance.
81. Designing for robustness.
82. Designing for testability.
83. Instituting design reviews for performance, robustness and testability.

### N. Automation of System Development and Maintenance

84. Automating the checking of code compliance with the coding standards.
85. Upgrading the programmers' workbenches with better tools, such as editors.
86. Automating the build process (to compile and integrate new system versions).
87. Automating the build verification smoke tests.
88. Developing a plan for more effective unit testing and integration testing, e.g., test-driven development.
89. Implementing, tightening or smoothing change control and version control.
90. Implementing complexity analysis and defect prediction.

### O. Performance Testing

91. Specifying the performance requirements and goals.
92. Developing a performance test plan template and a test planning guide.
93. Determining what to monitor and what to measure for performance.
94. Building the operational profiles.
95. Developing and using benchmarks.

96. Monitoring performance in live operation.
97. Developing guidelines for analyzing the measured performance data.
98. Developing guidelines for graphing and envisioning test information.
99. Developing guidelines for testing performance versus features.
100. Developing service level agreements and quality of service (QoS) goals.
101. Avoiding performance surprises by design reviews, early component-level testing, etc.
102. Developing guidelines for performance measurement data collection.
103. Calculating the test sample sizes.
104. Determining how to adjust (scale) the measurements from the test environment to the live environment.
105. Developing guidelines for scalability testing.
106. Estimating the number of performance measurement cycles.
107. Piggybacking performance testing on functional testing by adapting and re-using the existing functional test cases.

### P. Robustness Testing

108. Specifying the robustness requirements and goals.
109. Developing a robustness test plan template and a test planning guide.
110. Determining what to monitor and what to measure for robustness.
111. Monitoring robustness in live operation.
112. Developing guidelines for testing robustness versus features.
113. Implementing software reliability engineering.
114. Implementing software fault insertion.

### Q. Test Environment Management

115. Implementing better management and control of the test environment, such as installing diagnostic tools to monitor the state of the test environment.
116. Automating the set-up and configuration of the test environment.
117. Coordinating or linking complex test infrastructures.
118. Developing policies for managing the test environment and the test lab.
119. Developing guidelines for maintaining test productivity and utilizing test resources efficiently.

### R. Test Data Management

120. Developing the organization structure of the test case library or repository, and the procedures to manage this repository.
121. Establishing test case naming and numbering conventions.
122. Automating the generation of test data.
123. Automating the checking of database integrity.
124. Developing procedures for test case repository design and test case repository

maintenance.

### S. Test Case Development

125. Documenting the characteristics of an effective automated test scenario.
126. Developing test scenarios (specs of test cases to be automated).
127. Writing automated test cases.
128. Promoting test case re-use.
129. Test flow and sequencing.
130. Test case consolidation and minimization.

### T. Test Execution

131. Developing procedures for executing the test cases.

### U. Problem Management and Resolution

132. Developing guidelines for evaluating the test results.
133. Automating the test results gathering.
134. Automating the test results evaluation.
135. Implementing an effective problem reporting and tracking system.
136. Documenting procedures for developing problem reports and handling test failures.
137. Setting priorities and deadlines for fixes to be available for re-testing.
138. Assigning severity levels to problems.
139. Upgrading diagnostic and debugging tools.
140. Determining how to identify and resolve false test results.
141. Developing guidelines for building confidence and acceptance of test results.
142. Developing guidelines for handling inconclusive test results .

### V. Manual versus Automated Testing

### X. Test Project Management

143. Implementing the use of project management tools for test projects.
144. Automating the test project status tracking and reporting.

# *Appendix I: Avoiding Performance Surprises*

## APPENDIX I. AVOIDING PERFORMANCE SURPRISES

As we saw earlier, performance testing is often done very late in projects, when little time remains to react effectively and fix performance and robustness problems prior to the scheduled delivery date. Ideally, we would like to conduct performance testing before or during the system design phase, so that the feedback from testing can influence the design. This is not possible, because there is no system to test this early in the system development process, nor are the test facilities available.

With iterative development methodologies, where we design and built the system in a series of iterations, it is easier to use the performance measurements from one iteration of the system to project likely performance for he next iteration, but this extrapolation from version to version is often not very reliable.

What we need are methods to mitigate the surprises, even if we cannot eliminate the need for performance measurements on the fully integrated system late in the project. Some ways to minimize these last-minute “uh oh” discoveries in system performance testing are as follows. There is some overlap among these suggestions.

### 1. Early Design Reviews

The performance measurement team and the performance tuning specialists can plan to participate in the system design reviews, in order to identify likely performance bottlenecks at the design stage.

Designing systems for performance is often not done very well. In small, simple systems, the likely behavior of the system and its performance characteristics are fairly obvious to its designers. In large, complex systems, designing for performance usually becomes more important and also, unfortunately, much more difficult.

Design reviews naturally focus on feature validation. The designers and reviewers walk through the proposed design in a series of mental thought experiments, feature by feature, seeing if “we can get there from here”. We trace the processing of the system through the design model in order to confirm the system will work correctly and to find feature-level bugs in the design. In this series of mental experiments, the performance issues may be an afterthought or not considered at all.

So the presence of the performance testers in these design reviews helps focus the group’s attention on the performance implications of the design, and helps catch likely

performance problems in the design phase. We cannot guarantee this technique, and performance issues often are found later in testing and live operation which "fell through the cracks" during the review. However, the incomplete set of problems which we do see and minimize at this stage more than justifies the design review. Later in this book, I provide a checklist of questions for performance reviews of system designs.

### 2. Use of Prototypes for Load Testing

We can use early working or partially working prototypes of the system in performance measurement, as substitutes until the full system eventually becomes available for testing.

The unknown factor here is how close the performance of the prototype is to that of the real system. There is a danger of false confidence if the prototype does not exhibit the bottlenecks which are present in the real system. Since prototypes are "look and feel" shells with limited internal functionality and usually contain minimal built-in controls, such as error detection and recovery overhead, they often run much faster than their final delivered counterparts.

### 3. Performance Prediction

We can employ performance prediction tools to guesstimate performance from static models, i.e., without actually executing a test of the system. Therefore, we can use the performance prediction tools early, before the application and the test environment are available. Nevertheless, these performance modeling and prediction tools are frequently not very satisfactory, because they are expensive and can be unreliable. I'll explain how these tools work a separate section later in this book, entitled tbd.

### 4. Early Check-out of the Test Facilities

Even if the early performance measurements are considered so inaccurate that ee will have discard them, early testing is still useful to check out the test facilities. These early trials help the testers see if the automated load testing tools, test cases and test equipment are working, and to allow enough time to debug the test facilities.

### 5. Early Component-Level Performance Testing

A common difficulty with performance testing is that it occurs late in projects, after the fully integrated system has already undergone a feature test. This means that we do not identify scalability problems and bottlenecks until late in projects, when the design tends

to be “locked in concrete” and there is little time to react and re-code the slow portions of the software.

In an attempt to identify bottlenecks within the individual components early, we (or preferably the developers of the components) can perform a performance, load and stress test on each major component as it becomes available. This parallel approach works especially well in development projects which use an iterative approach to building component, or in projects where the developers are going to re-use existing components. These component-level tests usually need to be conducted using the component test drivers. Hopefully, the software engineers have developed these drivers already as part of the unit (component-level) testing. For example, we could utilize a component driver to stress a shared, multi-threaded component, by simulating the demand put on that component by ten concurrent threads.

The testers or the software engineers can use profiler tools to monitor the behavior of the components during this testing, so that the developers know where the resources, such as processor cycles, are being expended.

The component-level testing of performance ideally is closely coordinated with each component’s developer, and coordinated with the availability of early builds from the developers. Each developer will obtain early feedback about their component’s performance, giving the developers plenty of time to fine-tune and optimize component-level performance before the delivery of their components.

### 6. Early Trial Full-System Load Testing

The purpose of this early testing is *not* to measure the system’s response time and throughput and load. Often, the earliest measurements will be so rough (e.g., not within a target accuracy of plus or minus 25%), that they should be considered untrustworthy and discarded. Why, then, bother with early load testing? There are three reasons:

o Identify load showstoppers early, so that we can address them early. Major bottlenecks like deadlock in databases and missing Web pages, where the system waits indefinitely to try to access them, do not need precise response time measurements.

o To uncover application feature and processing bugs that only reveal themselves under stress.

o To check out the test facilities themselves, and allow adequate time to react and get them debugged and working.

## 7. Use of Simulators for the Test Environment

We can use simulators in place of missing or unavailable parts of the infrastructure in which the system will operate, such as computer chips which have not been built as yet, mainframes, databases and networks.

Simulators are generally software implementations of hardware devices in which we simulate only the necessary characteristics of the hardware in software, so the simulation is imprecise (building closely matched simulation models is very expensive and time consuming).

The types of simulators include environmental ones (which mimic the environment in which the system operates), functional simulators (which mimic the functions performed by the system we are testing or other systems which interface with it), and instruction simulators (these mimic the ability of missing hardware to execute the software instructions).

Simulation is also done for economic reasons, if it is prohibitively expensive to build the actual test environment, or for safety reasons if the live testing is going to hazardous (such as systems which launch nuclear missiles). Simulators are sometimes also called emulators.

## 8. Designing for Performance

One purpose of the design review process is to identify likely bottlenecks at the system design stage, so that we can alleviate them before we build, acquire or modify the software.

“How do you improve the performance of your software? Simple: just delete all the comments in the source code and recompile.” Do not take this advice seriously, but there is grain of truth to it. Often “sloppy” design and programming practices de-tune the system and cause bottlenecks and impediments to performance.

By “sloppy” design and programming we are referring to the tendency to make last-minute, supposedly minor tweaks to the source code and to the system structure. These tweaks presumably should not affect performance but often do: there may be unnecessary branches and convoluted paths inadvertently designed or built into the system. Even though deleting comments will not help, do not underestimate the power of simply walking through the design and the source code, looking for performance inhibitors.

## *Appendix I: Avoiding Performance Surprises*

There is another danger too. Tweaking to squeeze every nanosecond of performance out of code can render than code incomprehensible and extremely difficult to debug and maintain. We'll discuss this danger in the section entitled "System Performance Pressures".

Performance means speed, responsiveness and throughput. Designing for performance means identifying where most of the hardware processing is being performed (typically 90% or more of the processor cycles are expended in 2% to 10% of the software instructions), identifying wait states, removing bottlenecks and optimizing the code.

In order to assess performance based on the system design, the software components and their interconnections need to be already well defined. We then can estimate the likely usage of the software components using a simulation.

The usage profile identifies how often each software component is likely to be called. In addition, performing a performance assessment of the software design requires that we estimate the number of processor execution cycles for an average call to each software component. The number of execution cycles depends on the component's estimated size when it has been written (in lines of code), and expected mix of instructions within the component (floating point divides usually take much longer than register compares).

Tools are available to help assess system performance. Some tools work strictly from high-level models, though the more accurate ones (e.g., Quantify) require the source code, so they cannot work from high-level designs.

Some "obvious" ways to improve system performance may not be feasible. For example, consider an embedded system which is running too slowly. The first thought of the uninitiated is to "get a faster chip or add more memory -- hardware is cheap". (See the early discussion in the section entitled: "The Fat Server Solution".)

If the increment in hardware cost for a faster chip is only 25 cents per unit, but the organization plans to ship a million embedded systems per month, the cost is another $250,000 per month. You could hire another couple of performance testers with that kind of budget. (This remark is meant as sarcasm. $2.50 per month is closer to the mark.)

Some delays are inevitable in system operation, and if significant they must be minimized or bypassed where possible. Consider, for example, a system which depends on a dial-up modem. Modems are notorious for their latency, the wait time needed to

make a dial-up connection initially. A 56 kpbs modem sounds fast, until you consider how much time is spent waiting, both initially and then between each burst of data. Quadrupling the speed of the dial-up modem (224 kbps modems don't exist yet and perhaps never will), cannot help much with the latency.

In this situation, designing the system to minimize its dependency on the modem to the degree possible, could help boost performance considerably. Instead of many small back-and-forth data transmissions, perhaps the designers could bundle data together into a one-time large message for transmission.

Most of the techniques for improving system performance are "common sense", at least to experienced designers and programmers. The developers implement many of these techniques at the code level, not at the design level, i.e., by tweaking the source code rather than re-designing, but nevertheless these code-level performance improvements often have design implications.

For example, consider a software component that repeatedly derives the same value for a data field through a time-consuming computational process. To improve performance, the system calculates this derived value only once and then stores it, preferably in fast cache memory if it will be accessed frequently. So the software engineer will implement this improvement in the code, by writing source code to access the derived value rather than re-computing it.

However, the decision to store the derived value of the data field affects the design of the data tables and databases. It may lead to arguments with the data administrators about the purity of a database which contains derived, quasi-redundant data. Once the system has stored the value, there is a possibility many software components could use it. So the decision to store the derived value is a design-level decision, not one to be made by each individual software engineer who may need the value.

In addition, if many software components share the derived value, there is a possibility of interference among these components. A new component may need to be added to the design, with the exclusive job of provided access to the derived value and ensuring it does not become corrupted. And, of course, this new component carries a risk that it will become a bottleneck in itself.

## 10. The Performance Review

How do we review a system design to check if performance is likely to be acceptable? Following these steps will help ensure a fruitful design review.

# *Appendix I: Avoiding Performance Surprises*

*Have Timely Access to the Reviews*

Most testers are not invited to design reviews. A pity – testers see things that architects and developers can overlook.

*Review the System Architecture*

Preparation and familiarity are important: see the next point.

*Understand How the System Works*

First, we need to have a visible design. This means that the design has to be documented (preferably with clear diagrams), and sufficiently well understood by the reviewers that they can walk through it and perform mental thought experiments. A typical experiment is to follow any given event or message through the system.

The less detailed this model is, the less precise will be the performance review. If it is reasonably detailed (with major individual components identified, versus aggregations of components), though, the modeling of performance becomes very complex. We can use simulation languages like Simula or GPSS for these more complex simulation jobs. A caution: we need time to learn these languages, and we need more time and expertise to program and debug the simulation models.

Of course, if the design is detailed but the details are inaccurate or obsolete (designs tend to evolve), then the simulation results may be worse than with less detail.

*Review the Process Flow Model*

Ask if there are use cases. Preparation and familiarity are important: see the next point.

*Understand How the System will be Used*

We need to know who the users groups are and how they will use the system (operational profiles or usage patterns). We need some knowledge of the load on this design: what the mix of transactions or events will be, and in what volumes. We also need to know how each message, transaction or event will be processed: how it will flow component-by-component through the design.

*Walkthrough the Topology to Look for Bottlenecks*

## *Appendix I: Avoiding Performance Surprises*

Look for:

- Areas of heavy resource consumption.
- Areas that are timing-critical.
- Areas that are heavily used.
- Areas where resources seem to be unbalanced
  - over-capacity or under-capacity.

*Review the Operational Track Record of similar other Systems*

What similarities are there to the system being reviewed? Can the prior history be used to predict performance or the locations of bottlenecks?

*Review the Use of Shared Resources*

Depending on how they are shared, they can significantly help – or hinder -- performance.

Identify resources which will be heavily shared.

Identify resources which cannot be shared – one user at a time must lock a resource.

*Determine How to Minimize the Wait Times*

Identify likely areas where waiting will occur, either waiting for events to happen or for resources to become available. Most waiting is associated with I/O (input and output operations). For example, cache frequently accessed data in fast memory. Minimize the overhead spent in swapping data in and out of memory. Where waits are inevitable and the designers cannot easily control the latency, for example, in waiting for a modem to receive a response from a phone line, change the design in order to minimize the dependency on the device.

*Decide Where to Focus the More In-Depth Performance Review*

Focus first on optimizing the parts of the software where most of the processor hardware cycles are expended. This is likely to be within most heavily used and the most computationally intensive software components. Frequently used and resource-consuming code can be re-written and optimized to run very fast.

Then focus on areas where memory use is likely to be high, either semiconductor memory or hard drives. Ensure that sufficient memory is allocated to minimize

swapping.

*Review each High-Potential Component*

Based on the load, we can estimate the improvement potential of each component – how often it will be used, and approximately what resources it will need. We need to know this information about each component:

- How frequently will it be used, and under what conditions?
- What resources does it need to execute?
- Can or must these resources be shared with other activities, or must they be dedicated?
- About how long will it take to execute? (Tools like Quantify compute this.)
- If it is multi-threaded, how many parallel threads can be active concurrently?
- Does use of this feature or component precludes the use of other components?
- Has this component already been optimized?
- If not, to what extent is its performance worth improving (how much of a difference is optimization of this component likely to make?)
- How can its performance be improved? Minimizing the resource needs results in fast execution (e.g., lessening the number of processor cycles needed to perform a task). Performing tasks in parallel can cut the elapsed time. Software frequently contains many small inefficiencies of this nature which we can hunt down and remove. For example, is a variable being unnecessarily re-initialized (re-set to zero) before each use?

*Review the Efficiency of Computations*

Review computationally intensive software components to ensure that the most efficient algorithms have been used. It is common to have several algorithms which can compute the same outcome, such as a square root, but their computation speeds vary by a factor of a hundred. Areas which are often not seen as computationally intensive, such as database searches, often are culprits.

## *Appendix I: Avoiding Performance Surprises*

Ensure computations are not more precise than needed. Sometimes, adding another significant digit to the precision of a computed result doubles the computation time.

Do not re-compute frequently used values, unless the computation is virtually effortless.

Be aware of the relative efficiency of computer instructions, and look for faster ways to perform the same operation. For example, compare operations are generally much slower with mixed data types. In some combinations of hardware, operating system and compiler, it is faster to add a number to itself, twice over, than to multiply it by 3. In other environments, the exact opposite is true. It is helpful to be aware of the relative performance of the various instructions.

Check that an optimizing compiler has been used for the most resource-intensive software components.

Caution: optimizing for one particular platform is folly if the software might be ported later to other platforms.

### 11. Designed-In Flexibility for Tuning

I made the point before that most systems are de-tuned when they are originally delivered. This is not because of laziness and incompetence, instead it is similar to buying an off-the-rack suit and then having it fitted.

In most systems today, there and many different parameters which we can adjust and calibrate in order to optimize system performance. This implies that the performance tuners need flexibility to do their work.

### 12. Review Questions

Good questions to ask in a system design review are:

Performance Requirements

o What are the performance requirements?

o Have all the stakeholders and especially the end-user groups been consulted to determine if these requirements are acceptable?

## *Appendix I: Avoiding Performance Surprises*

o Are these requirements written in such a way that an independent test team can verify them?
  - Are they clear and specific enough to be measurable?
  - Have the conditions been defined under which these performance levels are to be achieved?

Loads or Demands

o What is the anticipated load from an external, black box view of the system?
  - Average?
  - Peak?

o How credible are these predictions?

Infrastructure and Architecture

o What are the target platform, operating environment and support infrastructure for the system?

o What is the system architecture?
  - What is the topology of the system and its infrastructure?
  - How are the components linked together?

o Have we identified each component of the system and its infrastructure (software, databases, network links and hardware)?

Capacity

o Have we identified the capacity of each component?

o Have we identified the circumstances in which the limitation of each component can be reached and surpassed?

o How do we calculate the capacity of the assembled system from the capacities and linkages of its components?

Process Flow

o How can we map this external load into the likely demand on each internal component?

- Is there a process flow model of the system?
- Do we have visibility into the design, i.e., can we conduct mental thought experiments where we walk through the design to exercise a feature or trace an event as it flows from component to component?

Other Prior Experience

o Are there similar other systems?
  - In which ways are they comparable?
  - In which ways not?
- Do we have the operational track records of these systems?
  - What was the level of satisfaction with the performance of these systems?
  - If there were performance problems:
    -- What caused them?
    -- How were they fixed?

Tuning

o What processes will people use to tune this system?

o What features, if any, have the system developers built in to facilitate this tuning?

o What flexibility do the tuners have?
  - For example, can they vary the size allocated to a queue?
  - If so, over what range can the maximum queue length be varied by the tuner, on site in the field?

## 13. Designing for Testability: Assessing Testability

The topic of testability, in the sense of determining whether a system is testable, includes considerations such as whether the system is too raw, buggy and unstable to bother starting to test it, whether the test infrastructure is ready, and whether the test team is ready. These particular items will *not* be addressed again in this section. Note that the term “testability” is sometimes used to mean the probability of software failure over time. This meaning is not being used in this section. (Personally, I do not like this meaning -- I think it confuses the situation.)

*The question examined in this section is how systems can be designed to improve their testability.*

# *Appendix I: Avoiding Performance Surprises*

## 14. The Characteristics of a Testable System

As systems become more complex, it becomes more difficult and eventually impossible to test them adequately unless they have been specifically designed to be testable. Many testers experience the frustration of testing systems for “hidden” internal behavior, and “looking under the hood” is a common part of testing. The point is that much of a system's behavior may be hidden and not directly observable from the outside, which severely limits the effectiveness of non-invasive black-box testing. For example, an internal buffer overflow may be extremely difficult to observe in testing or in live operation, unless a capability has deliberately been designed into the system to provide this information. In addition, in some types of systems, such as video games, the internal behavior is deliberately hidden in order to enhance the value of the system. (Where’s the fun in playing a game which is predictable?)

# *Appendix J: System Architecture Diagrams*

## APPENDIX J. SYSTEM ARCHITECTURE DIAGRAMS

This appendix provides examples of visual architecture models. These diagrams help us understand what the infrastructure is and how it works. It also helps us to hunt for bottlenecks, determine what behaviors of the system to monitor, where to collect data and how to interpret that data. Note that these diagrams represent the *logical* architecture, which is not necessarily how the design will be physically implemented.

### J.1 User-Oriented Architecture

This diagram groups the system capabilities together from the users' perspective. For example, in a departmental model we might have a dedicated server or cluster server for each major user group, such as Customer Service, Manufacturing or Warehousing, and these departmental servers share common databases of customers, products, orders, etc.

<<See separate file on System Architecture – it contains the diagrams>>

# *Appendix J: System Architecture Diagrams*

## J.2 Function-Oriented Architecture

This function-oriented view shows servers assigned according to the major functions they provide, such as a database server, a web server, a call center (voice telephone) server and a print server.

<<See separate file on System Architecture – it contains the diagrams>>

# *Appendix J: System Architecture Diagrams*

## J.3 Geographically-Oriented Architecture

This perspective allocates one area or portion of the page to each distinct location (e.g., London, Los Angeles, New York, etc.), and also shows how the different locations connect and interact. This type of diagram often is overlaid on a map, e.g., of the United States or of the world.

<<See separate file on System Architecture – it contains the diagrams>>

# *Appendix J: System Architecture Diagrams*

## J.4 Device-Oriented Architecture

The device-oriented view focuses on the likely physical implementation of the logical architecture. If some physical design decisions are not yet final, you may need to either ignore details or find a way to show uncertainties in your diagram. Drawing one big cloud-like figure on the page and placing a question mark at its center is too fuzzy to guide the performance test. But until the physical design is final some parts of the architecture diagram will effectively be little clouds, connected to other sub-systems on the page.

<<See separate file on System Architecture – it contains the diagrams>>

# *Appendix K: Performance Testing Work Plan*

## APPENDIX K. TEMPLATE OF THE PERFORMANCE TESTING WORK PLAN

### Part A: Test Preparation

This outline is intended to provide a starting point for developing test project work plans. This test work plan is developed within the context of a system release cycle, and will be coordinated with the overall development or maintenance activities and with the feature test plans. Some people are looking for a methodology for performance and robustness testing, a guide for how to do it. At the risk of sounding grandiose, this outline is a methodology, or at least the core of one. Other people believe strongly that there is no such thing as one right way to test system performance – the process is not cut-and-dried, and on any particular project competent people need to apply judgment and determine how to proceed. For the latter people, this is not a methodology, just an outline.

The typical performance and robustness testing project contains the following series of activities. Apply common sense in using this outline, as it needs to be adapted to fit your specific situation. Any particular project may not include all of these steps, and they may not happen exactly in the sequence shown here. Often these steps are not done in a simple linear sequence – as the test project evolves it can go through feedback loops and iterations of these steps. Tests of minor system upgrades tend be very different than ones of large new systems, so we often follow a different series of steps (a variation of this project outline) for these small applications.

The main activities in Part A are:

1. Gather background information on the situation.
2. Validate the test project need and feasibility.
3. Develop an understanding of the situation.
4. Develop the overall test approach.
5. Plan the measurement and load strategy.
6. Plan the test automation and testware.
7. Schedule and budget the test project.
8. Write and present the test plan.

The main activities in Part B are:

1. Prepare for the test execution.
2. Execute the test run and collect data.

3. Analyze data and evaluate the test results.
4. Present conclusions and recommendations.

**Project Outline**

1.0 Gather background information.

1.1 Understand the driving motivation and current impetus for performance and robustness testing.

1.2 Climb the learning curve.
Compile documentation about the system under test: functionality, architecture, support environment, usage patterns, support staff, etc.
Complete the information gathering questionnaire, or as much as is justified and feasible. (The questionnaire is attached.)

1.3 Review the project goals.
Identify the stakeholders and their critical success factors.
Review the stated goals and acceptance criteria for the new or modified system.
Determine if the goals and criteria are specific and measurable.
Determine if meeting the stated criteria is feasible.

2.0 Validate the test project need and feasibility (a “sanity check”).

2.1 Perform an initial impact assessment to assess feasibility and justification.

2.2 Review the management and client expectations for the test project.

2.3 Decide if these expectations are likely to be met.

2.4 Prepare a presentation justifying the testing project.

3.0 Develop an understanding of the situation.

Note: the steps in this section assume the testers are unfamiliar with the situation, and that it is worth their effort to climb the learning curve.

3.1 Understand the constituencies and vested interests.

3.2 Walk though the system under test with non-technical people (e.g., users) and

separately with technical people (e.g., the system architects).
The system's mission and functionality.
The organization's success factors for this system.
Process flow models, or other types of models to show how the system works.
The system's scope, boundaries and interfaces.
When the system will be ready for testing

3.3 Explore the system hands-on and on-line if it is available, or its predecessor if is not.

3.4 Understand the users.
Obtain their basic demographics.
Interview representative users to find their needs and opportunities.
Observe user work activities in their normal work places.
Build operational profiles if necessary.

3.5 Understand the testware needs.
Identify the resources which will be used by the system in live operation.
Determine how these will be represented in the test lab.
Review the existing test facilities and determine how they can be used in this test project.

3.6 Assess the baseline.
Review the data on the existing operations' service levels and user satisfaction.
Determine the adequacy of the existing data, and assess the need to capture additional baseline measurements.
Derive the message or story the baseline data is telling us. Determine the implications, if any, of this message for the testing project.

3.7 Understand the overall project context.
Determine how the performance and robustness testing fits into the bigger picture of the:

- System development, acquisition or maintenance project.
- Overall corporate culture and policies.

4. Develop the overall test approach.

4.1 Develop the testing project goals and validate them with the stakeholders.

4.2 Perform a risk assessment of the new system, to determine where the performance

and robustness issues are likely to be.

4.3 Develop contingency plans.

4.4 Determine the testers' additional information needs, if any, and gather this information.

4.5 Size the project.
Review the established test project goals to decide on scope and priorities.
Establish the test scope.
Establish the test priorities.
Determine how to measure test coverage.
Set the coverage goals for the project.
Define the test project entry & exit criteria.

4.6 Decide how structured versus exploratory the test project should be.
See the guidelines for structured versus exploratory testing.

4.7 Define the test results evaluation techniques.
What methods will be used to evaluate the test results.
What methods will be used to avoid false test results.

4.8 Coordinate the performance and robustness testing project with related activities on other projects.

5. Plan the measurement and load strategy.

5.1 Develop a measurement plan.
What to measure, why, when, where and how.

5.2 Determine what test loads to employ.

5.3 Determine what measurement tools to employ.

5.4 Determine what data collection and validation methods to employ.

6. Plan the test automation and testware.

6.1 Outline and justify the test automation strategy, and determine what techniques we will use in the automated testing.

6.2 Develop the test data acquisition strategy (test databases and test work loads).

6.3 Prepare the test environment.
Review the environment(s) in which the live system is expected to operate.
Decide what testware (test equipment, test tools, etc.) is needed.
Develop methods to adjust for the differences between the test vs. live environments.
Assess what existing testware from other projects can be re-used in this one.
Develop a test equipment acquisition or provisioning plan.

6.4 Develop the test cases.
Design the structure and format of the test case libraries.
Review the library format for maintainability and re-use.
Script (code) the test cases.
Validate the test cases: conduct peer reviews, and run and debug them.
Document the test cases.
Load them into the test library under version control.

7. Schedule and budget the test project.

7.1 Outline a work plan for the test project.

7.2 Estimate the project resource needs.
Qualified people.
Equipment and facilities.

7.3 Develop the project schedule and budget, and identify major milestones.

8. Write and present the test plan.

8.1 Draft the test plan in the required format.

8.2 Establish the procedures to update the test plan on an on-going basis.

8.3 Present and review the draft of the test plan, and incorporate revisions.

8.4 Present the test plan to senior managers.

8.5 Obtain approval to proceed with the test project.

9. Plan to use the test plan.

9.1 Describe the procedure by which we monitor the project's progress.

9.2 Decide how we compare progress to plan, and analyze the differences.

9.3 Describe how we update the test plan as conditions change.

Prepare to transition from planning to execution.

**Part B: Execute the Tests and Evaluate the Results**

Part B of this outline is less detailed than part A, because the specifics of the implementation depend on the decisions made in part A, the test planning.

Some of the steps suggested in Part B deliberately overlap ones from Part A.

1B. Prepare for the Test Execution

Prepare the test cases (test loads).

- Select and extract test cases from existing libraries.
- Adapt the test cases to fit this project.
- Develop new test cases as needed.
- Modify existing test cases as needed.
- Validate the test cases work, e.g., by trial runs.

Ensure that support processes and tools are in place and working, such as problem tracking, version control and project status reporting.

Set up and check out the test facilities.

Load the system version that we are testing.

Load other interconnecting or concurrently running systems (the background noise).

Load the test tools, and check they are correctly installed and configured.

2B. Execute the Test

## *Appendix K: Performance Testing Work Plan*

Determine the objectives of the next test run, refine the measurements to take and select the test load to match.

Run the test cases in the test load.
Exploratory tests; planned tests.

During the run, monitor the state of the test environment.

Capture the measurement data.

Review the measurement data for validity.
Is it trustworthy and usable?

3B. Evaluate the Test Results

Perform data cleansing and summarization.

Develop trend charts.

Derive patterns from the raw data.

Perform before-and-after comparison of patterns to discern the cause-and-effect of tuning.

Confirm or refute the hypotheses which were formulated during the test planning.

Form conclusions from the measurements.

Develop the recommendations.
Tuning and debugging.
Re-sizing of infrastructure.
System acceptance and readiness for use.

Re-test the system performance after the tuning and other modifications.
Develop or modify the test cases, the test environment and the data being collected, as makes sense.

Continue to loop through the cycles of tuning, fixing or right-sizing, followed by re-testing until the test goals have been achieved.

Prepare a final presentation of the test outcome, namely the findings, conclusions and recommendations.

Prepare the testware for re-use.

Present the findings, conclusions and recommendations.

Clean up the test lab, ready for the next lab users. Return borrowed equipment.

4B. Conduct a Post-Project Review

Lessons learned?

_____