Pro SQL Server 2005 High Availability

Allan Hirt

Apress®

Pro SQL Server 2005 High Availability

ISBN-13: 978-1-59059-780-4

ISBN-10: 1-59059-780-X

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Jim Huddleston, Dominic Shakeshaft
Technical Reviewer: Vidya Vrat Agarwal
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jason Gilmore, Jonathan Hassell, Chris Mills, Matthew Moodie, Jeffrey Pepper, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole Flores
Copy Editors: Jennifer Whipple, Kim Wimpsett, Ami Knox
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Elizabeth Berry
Indexer: Julie Grady
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section.

This book is dedicated to the memory of James Huddleston, who passed away unexpectedly during the writing and editing process of this book. He was one of my biggest champions at Apress, and you would not be reading this if it were not for him.

Contents

Foreword xix
About the Author xxi
About the Technical Reviewer xxiii
Acknowledgments xxv
Preface xxvii

PART 1 The Building Blocks of High Availability

CHAPTER 1 The Business of Availability 3

The Building Blocks of Availability 4
People 4
Training 5
Processes 6
The Cost of Availability 13
Defining Downtime 14
Nines 15
The Data Center Mentality 16
Planning the Data Center 17
Securing the Data Center 19
Power 21
Cabling, Networking, and Communications 22
Outsourcing Server Hosting and Maintenance 23
Technology 25
Summary 25

CHAPTER 2 Pay Now or Pay Later 27

The Genesis of a Solution 27
What Is the Business Problem You Are Solving? 27
Has This Been Done Before? 28
What Other Mission-Critical Systems Have Been Implemented? 28
Are You Biting Off More Than You Can Chew? 28
Are You Governed By Law or a Regulatory Body? 28
Do You Understand the End-User Requirements? 29

Money Changes Everything 29
Keep Your Friends Close and Your Enemies Closer 29
Service Level Agreements 31
Service Level Targets 31
Availability Service Level Agreements 32
Performance Service Level Agreements 33
Operational Level Agreements 34
Compromise 35
Time Is of the Essence 36
Support Agreements 36
Applications, Application Development, and Availability 38
Summary 38

PART 2 Always On: SQL Server Technology

CHAPTER 3 Backup and Restore 41

Understanding Backup and Restore Basics 41
Describing the Types of SQL Server Database Backups 41
Backing Up a Table 46
Mirroring Backups 46
Describing the Recovery Models 47
Understanding SQL Server Database Recovery 51
Backing Up and Restoring Full-Text Indexes 51
Backing Up and Restoring Analysis Services 52
Setting the Recovery Interval 52
Using Media Retention 53
Attaching and Detaching Databases 54
Querying Backup and Restore Information 54
Understanding Backup File Compatibility 54
Understanding Systemwide Options for Backing Up to Tape 57
Planning Your Backup and Restore Strategy 58
Understanding SLAs and Recovery 58
Dealing with Human Error 58
The “Need” for Zero Data Loss 59
Understanding the Link Between Disk Design, Database Layout, Retention, Performance, and Backups 59
Knowing Your Application 60
Backing Up to Disk, to Tape, or over the Network 61
Checking the Availability of Your Backups 61
Testing the Backups 62
Synchronizing Your Backups and Restores 63

Understanding Your Recoverability Paths 64
Managing the Transaction Log Through Backups 67
Implementing the Plan 68
Ensuring SQL Server Agent Is Started 69
Knowing Your Backup Requirements and Their Relation to Backup Frequency 69
Implementing Backup Security 69
Checking Database Status 70
Monitoring Backup Media and Backup Status 72
Using the Database Maintenance Plan Wizard 73
Backing Up SQL Server Databases 75
Restoring SQL Server Databases 84
Performing Piecemeal Restores 95
Backing Up and Restoring Analysis Services Databases 96
Automating the Retention Policy 103
Deploying Custom Backup and Restore Scripts 104
Summary 106

CHAPTER 4 **Failover Clustering: Preparing to Cluster Windows** 107

What Is Clustering? 107
Network Load Balancing Cluster 107
Server Cluster 108
Planning for a Windows Server Cluster 116
The Windows Server Catalog 116
Networking 120
32-bit and 64-bit Windows 122
Mixing Windows Versions 122
Disk Configuration 122
Security Configuration for a Server Cluster 127
Geographically Dispersed Clusters 129
Summary 130

CHAPTER 5 **Failover Clustering: Clustering Windows** 131

Step 1: Installing and Configuring Hardware and the Operating System 131
Step 2: Creating and Configuring the Cluster Service Account 132
Creating the Cluster Service Account 132
Adding the Cluster Service Account to Each Node 135
Step 3: Configuring Networking for a Server Cluster 141
Configuring the Public Network 141
Configuring the Private Cluster Network 145
Setting Network Priorities 148

Step 4: Configuring the Shared Disks 149
Step 4a: Sector Aligning the Disks 149
Step 4b: Formatting the Disks 152
Step 4c: Verifying the Disk Configuration 155
Step 5: Running the Microsoft Cluster Configuration Validation Wizard 156
Step 6: Adding the First Node to a New Server Cluster 166
Using Cluster Administrator 166
Using the Command Line 175
Step 7: Adding Other Nodes to the Server Cluster 175
Using Cluster Administrator 175
Using the Command Line 180
Performing Post-Installation Tasks 188
Configuring Cluster Networks 188
Resizing the Quorum Log 190
Creating a Clustered Microsoft Distributed Transaction Coordinator 191
Testing the Server Cluster 200
Summary 203

CHAPTER 6 Failover Clustering: Windows Server Cluster Administration 205

Remote Connectivity 205
Antivirus Programs and Clustering 205
Changing the Cluster Service Account Password 206
Disk Management for a Windows Server Cluster 211
Summary 224

CHAPTER 7 Failover Clustering: Preparing to Cluster SQL Server 2005 225

New Features of SQL Server 2005 Failover Clustering 227
Planning SQL Server 2005 Failover Clustering Instances 228
Number of Instances on a Single Windows Failover Cluster 228
Clustered SQL Server Instance Names 228
Clustering Analysis Services 230
Clustering Other SQL Server Components 230
SQL Writer and Failover Clustering 230
SQL Server Browser Service 230
Dependencies 231
Combining SQL Server 2005 and Exchange on the Same Windows Server Cluster 231
Security 232
Installing SQL Server 2005 Failover Clustering Instances Side-by-Side with SQL Server 2000 234

Installing Local Instances and Clustered Instances on the Same Hardware . . . 237
Disk Configuration . . . 237
Configuration Considerations for Multiple SQL Server Instances on the Same Windows Server Cluster . . . 238
Upgrading to SQL Server 2005 Failover Clustering . . . 245
Upgrading Analysis Services 2005 in a Clustered Environment . . . 245
Summary . . . 247

CHAPTER 8 **Failover Clustering: Clustering SQL Server 2005** . . . 249

Step 1: Ensure the Windows Failover Cluster Is Configured Properly . . . 249
Step 2: Create the SQL Server 2005 Failover Clustering Service Accounts and Groups . . . 250
Creating the SQL Server Service Accounts . . . 250
Creating the SQL Server–Related Cluster Groups . . . 251
Adding the SQL Server Service Accounts to the Cluster Groups . . . 253
Adding the Cluster Groups to Each Node . . . 255
Step 3: Rename the Cluster Resource Group . . . 256
Step 4: Install .NET Framework 2.0 . . . 256
Step 5: Install SQL Server 2005 . . . 259
New Installation: Setup . . . 260
New Installation: Command Line . . . 291
In-Place Upgrade: Setup . . . 295
Performing Post-Installation Tasks . . . 304
Installing SQL Server Service Packs, Patches, and Hotfixes . . . 304
Adding Additional Disks As Dependencies . . . 304
Changing the Affect the Group Property of the SQL Server or Analysis Services Resource . . . 308
Setting the Preferred Node Order for Failover . . . 309
Installing the Management Tools on the Other Nodes . . . 310
Removing the BUILTIN\Administrators Account . . . 313
Testing the Failover Cluster . . . 316
Upgrade Only: Changing the Service Accounts . . . 318
Summary . . . 320

CHAPTER 9 **Failover Clustering: SQL Server 2005 Failover Clustering Administration** . . . 321

Querying Failover Clustering Properties . . . 321
Using SQL Server 2005 Surface Area Configuration with Clustered Instances . . . 325

Starting, Stopping, and Pausing Clustered SQL Server Services 329
SQL Server Configuration Manager 330
SQL Server Surface Area Configuration. 331
Cluster Administrator. 331
Command Line. 332
Renaming a Failover Clustering Instance of SQL Server 334
Changing the Service Accounts Used by a Failover Clustering Instance 337
Changing the IP Address of a Failover Clustering Instance 339
Assigning a Static IP Port to a Failover Clustering Instance 340
Rebuilding master on a Failover Clustering Instance 342
Adding or Removing a Node 343
Using Setup 343
Using the Command Prompt. 349
Uninstalling a Failover Clustering Instance 350
Using Setup 351
Command Prompt 355
Changing Domains. 355
Changing the Domain with No IP Address Changes. 355
Summary 361

CHAPTER 10 **Log Shipping** 363

How Log Shipping Works 363
Best Uses for Log Shipping. 365
Disaster Recovery and High Availability 365
Intrusive Database Maintenance 366
Migrations and Upgrades 366
Log Shipping Considerations 367
Location of the Primary and Secondary. 367
Full Database Restoration on the Secondary 367
Sending Transaction Logs to More Than One Secondary 368
Transaction Size 368
Transaction Log Backup Frequency and Size. 368
Copy Frequency and Transaction Log Backup Location 368
Network Latency and Network Speed 369
Networks, Domain Connectivity, and Log Shipping 369
Log Shipping Between Versions of SQL Server 369
Code Page/Sort Order/Collation 370
Directory or Share Permissions 370
Synchronizing Database Logins 370
Objects That Reside Outside the Database 370
Log Shipping and Maintaining Consecutive LSNs 370
Log Shipping and Backup Plans. 371

Database Maintenance, Updates, and the Secondary . . . 371
Applications and Role Changes . . . 372
SQL Server Functionality vs. Custom Log Shipping . . . 372
Configuring Log Shipping . . . 373
Create the Backup Share(s) . . . 373
SQL Server Built-in Functionality . . . 386
Custom Log Shipping . . . 403
Postconfiguration Tasks . . . 403
Administering Log Shipping . . . 404
Monitoring Log Shipping . . . 404
Modifying Log Shipping . . . 408
Changing the Monitor Server . . . 408
Disabling Log Shipping . . . 410
Removing Log Shipping . . . 412
Adding Another Secondary to the Log Shipping Plan . . . 418
Manually Killing Database Connections . . . 420
Performing a Role Change . . . 421
Summary . . . 423

CHAPTER 11 **Database Mirroring** . . . 425

How Database Mirroring Works . . . 425
Transaction Safety . . . 426
Mirroring State . . . 426
Database Mirroring Modes . . . 427
Best Uses for Database Mirroring . . . 430
Disaster Recovery and High Availability . . . 430
Migration to New Hardware . . . 430
Reporting . . . 430
Database Mirroring Considerations . . . 430
High Performance Mode vs. High Safety Mode . . . 430
Edition and Version Configuration . . . 431
Location of the Principal, Mirror, and Witness . . . 431
Mirror Database Recovery Model . . . 432
Database Restoration and Configuration on the Mirror . . . 432
Server Sizing . . . 432
Disk Design and Performance . . . 432
Networks, Domain Connectivity, and Database Mirroring . . . 433
Code Page/Sort Order/Collation . . . 433
Security . . . 433
Distributed Transactions . . . 433
Transaction Size . . . 434
Transaction Logs and Database Mirroring . . . 434

Synchronizing Database Logins 434
Objects that Reside Outside the Database 434
Database Mirroring and Maintaining Consecutive LSNs 434
Database Maintenance, Updates, and the Mirror 435
Applications and Database Mirroring Failover 435
Configuring Database Mirroring 436
Step 1: Back Up and Restore the Database 436
Step 2: Set Up Database Mirroring 436
Step 3: Configure Network Load Balancing or a DNS Alias (Optional) 454
Administering Database Mirroring 454
Monitoring Database Mirroring 454
Controlling Database Mirroring 470
Altering the Database Mirroring Configuration 472
Removing Database Mirroring 473
Failing Over from the Principal to the Mirror 475
Full-Text Indexes 477
Redirecting Clients to the Mirror 477
Summary 478

CHAPTER 12 **Replication** 479

How Replication Works 479
Snapshot Replication 480
Merge Replication 481
Transactional Replication 482
Understanding the Replication Agents 485
Replication Considerations 486
The Application 487
Component Location, Network Latency, and Network Speed 487
Disk Performance and Sizing 488
Making Replication Highly Available 488
SQL Server Agent and Replication 488
Database Schema 488
Push or Pull Subscriptions 491
Security 491
Configuring Replication 492
Step 1: Configuring the Distributor 492
Step 2: Configuring the Publication 506
Step 3: Subscribing to the Publication 516
Administering Replication 524
Backing Up Databases Involved with Replication 524
Monitoring Replication 525
Summary 528

CHAPTER 13 **Making Your Data Available** 529

The Application 529
Making Third-Party Applications Available 530
Making Custom Applications Available 530
Getting In on the Ground Floor 532
Partitioning Your Data 533
Creating Objects on a Specific Filegroup 533
Partitioning Databases and Indexes with Transact-SQL 534
Partitioned Views 536
Data Dependent Routing 538
Database Snapshots 540
Creating a Database Snapshot 543
Database Snapshot Administration 544
Using Multiple Read-Only Databases 549
Summary 550

PART 3 **Administration for High Availability**

CHAPTER 14 **Designing High Availability Solutions** 553

What High Availability Technology Should You Use? 553
Comparing the SQL Server High Availability Technologies 554
Failover Clustering vs. Other Technologies 556
Log Shipping vs. Other Technologies 558
Database Mirroring vs. Other Technologies 558
Replication vs. Other Technologies 559
Combining the SQL Server High Availability Technologies 560
Failover Clustering with Other Technologies 560
Log Shipping with Other Technologies 563
Database Mirroring with Other Technologies 565
Replication with Other Technologies 566
Designing Your Solution 566
Performance 566
Sizing Processor and Memory and Purchasing Servers 567
Sizing, Purchasing, and Designing Disk Subsystems 569
Keywords 571
How Features and Functionality Affect Deployment 571
Designing with Disaster Recovery in Mind 571
Example Solution for an Existing Environment 572

Example Solution for a New Deployment 573
First Things First 573
Requirements 573
The Architecture 574
Planning and Deployment 574
Administration 575
Example Solution for Disk Configuration for a Failover Cluster 576
Requirements 576
Planning and Deployment 577
Summary 578

CHAPTER 15 24x7 Database Administration 579

Testing and Change Management 579
Installing and Configuring SQL Server 579
Installing SQL Server 579
Configuring SQL Server Instances 584
Setting Memory Values 587
Configuring Databases 588
SQL Server Security 594
Securing the Instance 594
Securing the Application and Databases 602
Dedicated Administrator Connection 603
Monitoring SQL Server 604
What Should You Monitor? 604
Using a Monitoring Application to Monitor SQL Server 612
Using Performance Monitor to Monitor SQL Server 613
Using SQL Server Dynamic Management Views to Monitor SQL Server 617
Getting Notified of Problems 618
Attaching and Detaching Databases 638
Detaching a Database via SQL Server Management Studio 640
Attaching a Database via SQL Server Management Studio 641
Using SSIS to Transfer Logins and Objects 645
Abstracting a Name Change During a Server Switch 658
Using Network Load Balancing 658
Using a DNS Alias to Abstract a Server Name Change 670
Summary 672

CHAPTER 16 24x7 Database Maintenance 673

Performing Database Maintenance 673
Creating SQL Server Agent Jobs 673
Performing Routine Maintenance 686

Performing Server and Instance Maintenance 690
Handling Physical Disk Fragmentation 690
Performing Maintenance to the Server Itself 690
Disabling Automatic Windows Updates 691
Applying a SQL Server 2005 Service Pack 692
Importing and Exporting Data 703
Using bcp to Import and Export Data 703
Using SSIS to Import and Export Data 704
Summary 712

CHAPTER 17 **Disaster Recovery** 713

Expect the Unexpected 713
Preparing for Disaster Recovery 713
Data Loss 714
Plan in Advance 714
Data Center Access 715
Have More Than One Plan 715
Documentation and the Run Book 715
Staffing and Chain of Command 718
Supplies and Contingencies 719
Test the Disaster Recovery Plans 720
Check Your Support Contract 720
When Disaster Strikes 721
Assessing the Situation 721
Contacting Support 721
Implementing the Plan 721
Maintaining Your Cool 722
Shadowing and Documenting 722
When the Dust Settles 722
SQL Server Disaster Recovery Features 723
Summary 724

INDEX 725

CHAPTER 1

The Business of Availability

We live in a connected world that is seemingly *on*—or more accurately, *online*—24 hours a day, seven days a week (otherwise known as *24x7*). Whether you are checking e-mail on your cellular phone, sitting in a coffee shop browsing the Internet, or doing some shopping via a wireless connection, the need and desire to access vast amounts of information has never been greater in any other point in modern history. The world is literally a click away.

However, this demand for information anywhere at any time means that somewhere in the world a set of servers must be up and running 24x7 to support access to data. End users may take it for granted that they will always be able to access information; but you know that achieving around-the-clock availability for systems is by no means a trivial task. The question ultimately arises—when the system goes down, what is the end result? Is it lost revenue? The possibility of a lost life? Something else? Unfortunately, there is always a cost associated with downtime, and some costs have more dire consequences than others.

World events such as the terrorist attacks on the U.S. Pentagon and the World Trade Center in New York, Hurricane Katrina in New Orleans, and the tsunami in Asia are excellent indicators of how devastating catastrophic events can be. No amount of preparation could have prevented these tragedies, but they demonstrate that planning for the worst-case scenario is absolutely necessary, and that minutes versus days and weeks makes a difference on many levels—economic, social, technological, and financial.

The ripple effect of September 11, 2001 (despite the human tragedy, which will never be erased), could have been much worse since it struck so many businesses. Many of those businesses that were affected were in the financial industry or linked to it and could have disrupted the entire financial industry on a global basis for an extended period of time. Many of the companies maintained contingency plans with facilities elsewhere, which ultimately allowed business to get back to relative normal sooner rather than later; otherwise, that ripple effect would have been crippling and may have taken quite some time to recover from. This is also known as *business continuity*, which is what you want for your business: to go on. Aside from a tragic event like 9/11, a single disk failure in one server at your company could cause downtime that could cause a catastrophic impact on your business.

If you think business continuity is just an IT problem, it is not. There are so many other aspects to a normal routine being able to continue after a disaster. Some industries mandate that there are plans in place, such as the health-care industry. Laws sometimes require continuity. Some examples can be found at `http://www.gartner.com/DisplayDocument?doc_cd=128123`. At some point, with no cash flow and no way to get back in business, businesses fail and go bankrupt.

Some people naïvely assume that achieving availability is only a technology problem. It is as "simple" as spending some fixed amount of money, acquiring hardware, setting it up in a data center, hooking up some disks, and letting it run, right? Wrong! If it was that easy, you would not be reading this book. Availability is *business*-driven, not technology-driven, since there is always an economic reason why you actually need to achieve some form of availability. Technology only represents the physical aspect, or implementation, of the availability chain.

How you achieve availability is multidimensional: are you isolating against a "local" failure, or a more global problem? A local failure—when a problem occurs in your data center—is where *high availability* comes into play. For example, if a memory chip goes bad in one of your primary database servers, having hardware that supports the ability to replace it on the fly would be a way to mitigate a local failure and possibly maintain full availability (with reduced performance, of course) of the database system to the application or applications it is serving.

If the primary data center burns down, or the failure is less isolated, it becomes *disaster recovery*. While disaster recovery is closely related and tied to high availability, it is a completely different exercise to plan and account for. Another way to think about high availability versus disaster recovery is that high availability tends to be based on some sort of measure during what you consider "normal" business, whereas disaster recovery is measured on something "abnormal." Disaster recovery is discussed in more detail in Chapter 17.

The easiest way to think of both high availability and disaster recovery is to equate them to insurance policies; if you never have a problem, they will not be invoked. Upper management may even complain that money was wasted. That makes sense if you think about it; it is hard to quantify the pain if you are not actually feeling it, but it only takes one outage to convert the nonbelievers to the religion of availability.

Playing the what-if game may seem pointless to some, but the reality is that it is a matter of *when* you will have a problem, not *if*; so if you put the right mix of people, processes, and technology into place, the impact felt should be minimal and the end result should be one where you will not suffer much.

The Building Blocks of Availability

Even with technology in the mix, availability first and foremost stems from people and processes. This may be the last thing most of you want to see taking up space so early on in a technology book, but the fact remains that it is the truth. Without people, you cannot plan, implement, and manage; without processes, things will fail faster than the blink of an eye. Chaos and flying by the seat of your proverbial pants is a surefire way to lower your availability, not increase it. Investing in both people and processes should be fundamental to any organization actually wanting to attain high availability, not one that just pays it lip service.

People

It is not an exaggeration to say that without good people, nothing happens—or more accurately, nothing happens right. Giving an unqualified or incompetent person a job is not the way to hire your staff. Having the right people in the right place with the right experience can potentially mean the difference between minutes and days of downtime.

I am often asked if a person who has the certification of Microsoft Certified IT Professional: Database Administrator, MCSE, or MCDBA is the best candidate. Like many things in life, it all depends. If someone with the relevant experience has the certification, he most likely went through the process to make himself more marketable (since certification is perceived as showing you know what you are doing) or to further validate and make official what it is he does on a day-to-day basis. The certification may mean little to him, other than maybe getting him a higher base salary because someone else values it.

If someone who has just graduated from college and has never used SQL Server before goes through Get Your Cert Fast Inc.'s cram course on what to study, or just attends a few training classes, he or she is by no means the best candidate for the job. Nothing can substitute for actual real-world experience in designing, planning, implementing, managing, monitoring, and maintaining systems. Certifications are valid; but never make your staffing decisions based on that alone. Putting

your production environment in the hands of the Microsoft Certified IT Professional: Database Administrator with no experience is akin to playing Russian roulette: you may hit the empty chamber and the person will work out, or you may get the loaded chamber where his or her inexperience may hurt you. How much risk can you assume?

Ideally, the best candidates will ultimately have that perfect mix of book learning, relevant experience (maybe not in the number of years, but in the actual experience they have gained), and experience working with others who have mentored them and helped them absorb traits such as strong communication skills, as well as the elusive ability to troubleshoot and prioritize, all while reacting to any emergency thrown their way.

More relevant to the topic at hand, do you want some neophyte who has never faced adversity or true pressure running your disaster-recovery procedures? Keep in mind that there is a different mindset of just maintaining your servers on a day-to-day basis than there is with knowing that you need to have business continuity. In more cases than not, it is always better to hire a candidate who has 12 years of relevant experience with no certification who knows the difference, and then hire your certified junior person and have him or her soak up the experience of the veteran. There is nothing worse than having a young hotshot who thinks he or she knows it all, because that brazen overconfidence based on some minor experience or book knowledge can be a dangerous thing.

You are looking to run a marathon, not finish the race winded. Both types of workers have a place in your organization. But many companies do not want highly paid and experienced senior-level administrators doing day-to-day work when they can get someone at a third of the cost. You have to realize what end of the spectrum the people you are hiring are on and where they will fit into your organization. To maintain availability, you need local expertise you trust, and quite frankly, good DBAs are not easy to come by.

The bottom line: hire quality people and your chance for success goes up exponentially. The same applies to network administrators, storage engineers, and anyone else you may hire.

Training

Training represents a way for companies and people to grow their skills in a formalized environment. Unfortunately, many implementations of the corporate training program leave a sour taste for most of the users and nonusers of the system put in place. Most companies are supposed to allow their employees a yearly allotment of training and development, but all too often the employees who most need the training are never able to attend because they are busy running the business. It is very rare to encounter a company enlightened enough to hire the "extra" worker necessary to allow the senior technologists running the day-to-day IT infrastructure to train others on the system in case they're unavailable, let alone learn something new.

Putting such an infrastructure in place allows these employees to not only attend classes and conferences, but to also come back and present what they have learned. Staffing levels need to be designed around support for continuing education commitments. Ultimately, the business needs to ask itself a simple question: what is the cost to the business in the future if that DBA's skills lapse? It is a double-edged sword.

In a mission-critical environment with extremely high availability requirements, it is absolutely crucial that everyone involved with managing the systems stays up-to-date with their skill sets. If at all possible, they should stay ahead of the curve and be allowed to look at beta releases of software. This requires a serious commitment on the part of the employer, but one that should pay great dividends.

What I often see happen is that individuals who work a zillion hours per week and barely have time for family and friends, let alone training, spend their own time on learning it. This is a slippery slope because then the business can "expect" that it is acceptable. It is not. Training should occur on the clock. Burning your employees out and destroying any hope of a work/life balance lowers morale and, ultimately, retention rates. Training should always be a positive

benefit for your employees and a reward (maybe a week out of the office for a change of scenery) for all the hard work they put in day in and day out.

If sending employees out to training or conferences is not a possibility, remember that training can also come in other forms. Mentoring and activities such as "brown bag" lunches to share knowledge and experiences can go a long way. Many software companies offer on-demand training over the Internet. For a junior employee, there is no better learning than on-the-job with mentoring. Formal training has its place, but you do not learn how to manage a 24x7 environment by sitting in a classroom. It is always an eye-opening experience the first time you are involved in a server-down situation; and for junior employees to be involved in the recovery process or to observe it will only further prepare them to be able to do their jobs.

Another method of training that works well is to have peers and senior-level technologists create lab exercises for the more junior employees. This requires having some dedicated hardware. For example, set up a specific solution (such as a cluster), and then break it on purpose. Have the trainees then go through the process of troubleshooting and see whether they can fix the issue, find all the issues, and see how long it takes. These are the types of skills that could prove crucial in an outage.

Not all things that are educational in a corporate environment need to be on a one-to-one or small-group basis. For example, one of the things you will need to do is hold "fire drills" where you test your failure scenarios. Going through the drill, you learn quite a bit about not only what is possibly broken in the process and needs to be fixed, but whether people truly know their roles and whether everyone is communicating as expected. The business needs to learn and grow as well.

■ **Note** Do not forget to develop the soft skills of your employees. Maintaining their technical skills is ultimately important and critical for the tasks at hand, but knowing how to communicate is an equally important skill to have. In a disaster recovery situation, cooler heads will prevail; and if there is a breakdown of command and communication, you will only lengthen your availability outage.

Processes

Along with having the right people, there is another key piece to the puzzle: process. Process is something many IT workers have acquired a hatred for over their years in IT. Ironically, the ones who curse and fight it at every step have the most to gain from it. If you delve into the issues around the "annoyance" of processes, more often than not it is not the actual processes themselves, but their delivery.

Consider the following questions:

- Was the underlying purpose for the process clearly explained?
- Was the purpose presented in a way that made it clear and compelling?
- When the process was created, was I consulted for input? And was my input included in the final process when relevant?
- Are there obvious exceptions to the process that the process left "open" and did not attempt to solve?
- Are there steps of the process that ultimately hurt the business from the point of view of the members of my team?

When these types of surrounding concerns are properly addressed, processes can become much more of a friend than an enemy. These processes can help by allowing workers to find out how other workers resolved a problem. This type of information can prove critical in an outage where the problem has been seen before. Reinventing the wheel is not something you ever want to do.

The secret to a successful set of implemented policies is enabling the proper communication to ensure the process works for everyone. There is nothing more damaging to a company than poorly defined or thought out (or nonexistent) processes that are ignored or used in such an inconsistent way that they are ineffective. That definitely leads to outages, chaos, and confusion. In a worst-case scenario, management may react in a way that seems contrary to what is needed, which may lead to critical failures. Management may make this type of decision based on *published* policies versus what is actually *practiced*. The two should stay in sync. Unfortunately, the policies that are actually practiced may have been developed as a result of resistance to the published policies, or they may have evolved and adapted to the working conditions through necessity (not unlike Darwin's theory!).

Even the most skilled and qualified of workers cannot combat processes that are that horribly broken. I have no hard statistics to back up my claim. I am only drawing upon my experiences and those of others I know. It is a pattern I have seen over and over again in my travels and experiences in all sorts of environments and in different roles. The size of the company or IT department does not matter, but it is safe to say that the larger the environment, the more glaring and magnified the problems.

The three biggest types of process breakdown seem to occur around communication, change management, and testing.

Communication

Arguably the No. 1 factor that contributes to downtime is communication, or lack thereof. While this is obviously in part a people problem and could be discussed in that light, it is also related to process and how things work in your environment. Communication is the central hub of a well-oiled organization. Poor communication inhibits availability and creates barriers to success.

The communication issue is multilayered and complex to deal with if there are problems within your organization. Why? There is communication within a specific team, and then there is communication with other groups, which can even be extended to include contact with customers or end users. Both types of communication are crucial, especially when you are in a 24x7 environment.

Intergroup Communication

For better or for worse, most environments (mostly larger ones) have a clear division of powers in an IT organization. This has some obvious benefits, such as concentrating areas of expertise within a single group instead of having it dispersed; but in some cases it can complicate matters. For example, many companies have separate groups of network administrators, system administrators, and storage administrators, in addition to a complement of DBAs and other types of administrators. These groups are usually started with the best of intentions to serve a specific need since it is easier to grow domain expertise with a focused group of individuals. Where it breaks down is when some things actually need to get done and some form of compromise must be achieved. Each group may have its own work style, policies, procedures, standards, and other preferences that will conflict with another group's. In the end, too many hands in the pot can make something that should be simple become something that takes an act of Congress or Parliament to approve due to partisan posturing where everyone stands their ground and no one is willing to compromise. Everyone needs to work together as a team, not as little fiefdoms trying to one-up each other. For example, if you have different groups (storage, backup operators, DBAs, and possibly even a few others) involved in the backup (and eventual restore) process, it is unrealistic and time-consuming if everyone points to the DBAs as being the only group responsible for restoring the backup in the event of a problem. The reality is that the DBAs are absolutely necessary, but the central IT group owns the network-based backup solution backing up the SQL Server databases, and the backup operators have already moved the archive tape

offsite. How can the fingers point to the DBAs as the problem if it is taking hours, since it will take quite a few other groups to do their jobs before the DBA can even think about restoring the database?

If you cannot work together as a team under normal circumstances, how can you expect to do it in a server-down situation? When the dust settles, everyone has to play by the same set of rules as determined by your organization, and having standards that everyone must adhere to (despite the technology being different) should put everyone in their respective places. You are all managing the same goal. It stands to reason that problems are only magnified when the pressure is on, and most likely you will experience a lot of confusion and pain. Cross-group communication strategies that work need to be devised and implemented.

Intragroup Communication

Even within a group, such as the DBAs or the network administrators, there may be no unity of purpose. This usually occurs due to assumptions, poor documentation, or lack of vision in the group as to its overall goals and deliverables. There is no way a single group can be effective if there are a million different agendas going on and people not communicating properly with one another. If the DBAs cannot do their job, how can they expect to meet the availability goals before them? A good manager will foster a positive environment where all of the petty, catty behavior is left at the door and everyone has a hand in (and can be rewarded by) the success of the group as a whole.

Individuals are important, but what does it matter if Bob takes care of the backups when Sue is ill? That is what should happen, but often does not. Poor intragroup communication can often lead to the blame game where someone wants to pass the buck ("it was not my responsibility"). That is unacceptable in IT shops small and large. Every individual should be responsible; otherwise, what is the point of hiring them?

Setting End-User Expectations

End-user expectations are viewed in a somewhat different light in Chapter 2, but for this discussion, the purpose is simple: if there is going to be downtime, communicate it to all who need to be "in the know." For example, if you are going to be taking a maintenance window for your servers and this affects, say, an online store where people normally have the ability to shop around the clock, make sure that a friendly message is put up on the web server that says something like the following:

> *Thank you for visiting our online store. We are temporarily down for maintenance and will be back up for your shopping pleasure at 04:00 GMT.*

This may seem like an obvious thing to do, but it really is not to some. Think of it another way: if you as a consumer go to a web page and get an error, you will try to visit a few more times and at some point move on, right? It is human nature. While that may be fine for you—you found what you want elsewhere—what that means is that the online retailer may have lost a customer, possibly permanently. Lost business is not a good thing. Putting up user-friendly messages lets people know what is going on, that you are still in control, and the situation is being handled. On top of it all, you may retain your customers since you are telling them when they can come back, not unlike a store posting its hours outside the door.

You should also treat your internal business customers the same way you would treat external ones since proper notification allows those affected to make the appropriate plans. If you do not, all of a sudden you may have some manager breathing down your neck because his or her application, database, web server, and so on, was unavailable when he or she needed it. When managers are unhappy, you are unhappy. So if you need to take a system down to apply an update, give business users a heads up with decent lead time so they can plan ahead, and definitely let them know how

long the outage will be. I cannot tell you how many times in the past ten years I have been onsite with a customer and suddenly the resource we are accessing is unavailable. Why? Someone rebooted the server and did not let anyone know, or some other similar event happened.

What this all boils down to is providing good customer service whether your customers are sitting in their houses buying widgets, or sitting in a cubicle in your office building trying to do their jobs. From your perspective, the attitude you must have is that what you do as an individual affects much more than yourself; one action you take may affect an undetermined number of people.

Testing

A failure in many environments—and it is glaring in the change management process—is testing. *Testing* is not just testing the application itself, but everything involved in the whole ecosystem that makes up the solution. Arguably, testing the application is most important. For example, I cannot tell you how many times at this point I have heard "So we upgraded from SQL Server 2000 to SQL Server 2005 and we are seeing (insert issue here, most likely related to performance)." I then always ask if they tested the application with SQL Server 2005 before upgrading, to which more often than not the reply is no. "We assumed it would work just fine since the upgrade went smoothly." Sometimes they actually do some testing, but none of it is under actual or simulated production loads, or with data (both in size and makeup) that is representative of production.

The bottom line is that one should never assume that things will "just work" since an oversight can become costly. You do not want to find out that what you thought would be a simple two-week effort based on an assumption will actually be a four-month effort.

Coming from a quality assurance background, I know how important testing is and how much it is impacted by many factors. If your development cycle runs long but the ship date cannot change, what gets squeezed? Most likely it is the testing time, or the amount of coverage within the application that is tested is reduced to get the product out the door. I understand the bottom line, but if the product blows up in production, causing you to spend weeks or months patching it just to get it usable, is it worth it? It not only causes problems with future development projects, but there are most likely large numbers of dissatisfied customers who will change back to the old version of the product or find another vendor because your new offering is unreliable and does not meet their business needs. You cannot afford either of these occurrences. Plan the appropriate amount of time for testing and building quality into the product.

You may not think of it this way, but testing is one of your first weapons for achieving availability. Will things work as advertised or promised? If the application does not work, it could mean downtime due to retooling. Too often, the focus is on testing: what has changed instead of what is important to the business. Focus on ensuring that your application's core deliverables are not affected, instead of trying to test underlying functionality that you may or may not be using. This customer-focused testing will pay off in the long run. To be quite honest, it is not, nor should it be, the IT department's job to first figure this out. By then, it is too late. IT most likely does not know much, if anything, about the application itself yet is tasked with supporting it. This means that if you are deploying your back end with certain technology, it is imperative at the testing stages to test to see whether the application will work with the technology in question and what will happen should a disaster strike. This also means the IT staff and the application developers and support staff must be in contact from the beginning. With third-party applications, you may not have the ability to do as much extensive testing and customization as you would with your own applications. This does not absolve you from testing them properly in your target environment.

For example, if you are deploying mirroring at the database level (a feature of SQL Server 2005 that will be discussed in Chapter 11), what happens in the failure of the primary server and the switch to the mirror? Not only will you learn about application behavior and how you may need to change the application to deal with the server change, but administrators or DBAs can also test their processes to ensure they work.

Change Management

Change management is the process where updates to applications, such as schema changes, or fixes, such as service packs, are applied in a way that causes minimal or no interruption in service to the business and/or end users. Production should never be the first place you roll out a change. Very few companies have an operationally mature change management process—if there is even a change management process at all. Change management is never important to a company until it has an outage and realizes that it could have been prevented with a proper change management process. The excuses are always plentiful (and sometimes entertaining):

> "It worked on my machine."
>
> "We do not have the budget for test or staging servers, so we just apply changes directly to production and hope for the best."
>
> "We assumed it would just work since the patch came from our vendor who we trust."

Do any of these apply to you now? If so, how many of these excuses have you encountered after an update fails in production? I would venture to guess that most of you have heard excuses at some point in your career. This happens every day somewhere in IT Land. And outages due to updates to servers and applications, for the most part, can be prevented. The lack of change management is a gap in preparation for availability and can cause you tremendous grief and aggravation. Change management needs to be a way of life.

■ **Note** To assist you in change management and other aspects of your organization, there are existing frameworks you can refer to. The most well-known is called Information Technology Infrastructure Library (ITIL). Microsoft has a few subsets of ITIL—the Microsoft Operations Framework (MOF), `http://www.microsoft.com/technet/itsolutions/cits/mo/mof/default.mspx`, which is geared toward IT professionals, and the Microsoft Solutions Framework (MSF), `http://www.microsoft.com/technet/itsolutions/msf/default.mspx`, which is geared more toward development. Both are similar and are meant to be a set of best practices to guide you in continually managing your environments, as change never stops.

Set Up Multiple Environments

One of the key things that you must do to ensure that you have a successful change management process is to establish other environments, in addition to production that will create separation and, in certain cases, redundancy. Those environments are development, testing, and staging.

Development and testing are pretty self-explanatory, but are often not implemented in an optimal way. In a best-case scenario, the development and testing environments will be isolated from one another. Some companies may implement solutions such as virtual machines and allow each developer to have his or her own environment, but all changes are rolled up centrally. All change processes and technologies deployed in development and testing should prove out what will eventually be performed in production. What you have in development and production in terms of environments should in some way reasonably resemble what production will look like. How can you expect to be successful if what you are developing in no way, shape, or form looks or acts like the end target? For example, if the intent is to have your target application work on a clustered system, you would simulate a cluster (or have a real one) in your development and testing environments. Developing for clusters is similar, yet different, than developing for a stand-alone server. Never assume it is someone else's problem.

Depending on your solution in production, having a facsimile in development, testing, and/or staging could be an expensive proposition. In most places I travel, development usually does not get this setup. It is unfortunate that the company does not see that failing to invest early

leads to problems during implementation. For example, in development, the solution and all of its components are configured only on one server. In production, they will be spread out over multiple servers: a web server, an application server, a middleware server, and a database server. When the solution is deployed, much to the shock of no one, the installation fails because the solution is designed to work as a one-box solution only, without taking into account the complexity of decoupling all of the components. I know I have seen this in action, have you?

Staging is the ultimate key to IT success. This is usually an environment that is an almost, if not exact, copy of your production that no one but IT uses. This is the last place where any changes can be tested before rolling them out in production. I understand that most companies can barely afford what it takes to have one copy of production for high availability or disaster recovery, let alone more. But how many times have you seen where a production environment needs to be rolled back to a previous state because no testing of a fix has been done anywhere else?

If you have a staging environment, in the event that your production hardware fails, and assuming that clients can be redirected to this other environment and data is somehow synchronized, it could possibly be used for some sort of short-term disaster scenario (although this is discouraged; you should have a proper disaster recovery solution). This is an example of how investing in a staging environment can be viewed as more than just some servers that are sitting there and have no other purpose than to test changes that may only happen twice a year.

Beyond rolling out application-based changes, a staging environment is most important for proving out operating-system and application-specific patches such as hotfixes and service packs (see Chapter 16). Take the example of a failover cluster, which is a specific implementation both at the hardware and software levels. Rolling out patches to such an environment is not necessarily the same as doing it to a stand-alone server and may have its own gotchas associated with it. Applied improperly, you are looking at downtime, and at worst, full system recovery in certain cases. Is that something you want to do for the first time in production? Once again I will say this: buying systems is not cheap. But how much is your uptime worth to you?

Implementing Change Management

Change management is hard to do if you have never done it before. It requires quite a bit of discipline. Whatever you employ for change management *must* be flexible enough to handle emergency situations where things need to get done immediately. I have seen where change management can also get in the way during server-down situations, while people are waiting for upper management to sign off on something. This is a simple equation:

Minutes going by = downtime = consequences

Implementing basic change has four main aspects:

- Locking down the production environment so that changes can't be made or can't be made without proper approval
- Creating an approval process for changes that includes test sign-off for those changes
- Creating a test process that provides the business confidence that testing reasonably covers what is done in production
- Creating a process to track proposed changes and their status, as well as their results, in terms of success or needed improvements over time

Hopefully the DBA is involved with all aspects of the database implementation, from the application on up through the implementation plans at an early stage in the process. If not, it is a change management problem waiting to happen since the DBA will have no clue how things work or why things are being done the way they are and will likely have to question changes for the wrong reasons.

Most environments will have one group of DBAs for development and testing and another for staging and production. The information flow needs to be uninhibited between the two groups and, arguably, DBAs should be rotated so that they are always kept fresh and have the opportunity to see all aspects of the business.

Remember that changes to the servers and components under SQL Server—from development up through production—impact the availability of your SQL Servers.

Here are some best practices for change management as they relate to highly available database environments:

- Make backups of all databases (including the SQL Server system databases) prior to attempting any change. You may need them if the application of the change or update fails and you need to revert to a previous state.
- Make backups of all databases (including the SQL Server system databases) after a successful update. These will be your new baseline full backups of the databases.
- Change management should be based on forms filled out by the appropriate parties—either electronic or paper-based. The form should at a minimum include what the change is, why it needs to be done, who will be performing the task, the steps to execute the change, and what the plan is if something goes wrong; and it should have places for signatures (real or electronic) of people empowered to approve the change. This provides accountability for all involved and a way to look back if something goes wrong later so you can track what changes have been done.
- Application developers should never assume that applying changes are only an IT problem. Developers should package all application changes (including database schema changes) in easy-to-deploy packages that can be rolled back without having to reinstall the entire application or server. If they do not, have them witness how painful it is to apply the changes and see if they like it. I will bet that they will go back and change their code to be more IT-friendly.
- Use version control for any code written, as well as key components such as baseline databases. I have been in too many environments in my career where at some point it is discovered that the wrong version of a script was run against the database, or the wrong DLL was deployed in a build, but due to lack of proper version control, there is no way to go back and deploy a previous version of whatever went wrong, or even build it since the source tree only contains updated source code and the old source that built the DLL no longer exists. This is also an issue when a problem is introduced in a later version and regression testing needs to be done, and the only way to get older versions is through source control.
- When testing changes, record how long it takes to apply the changes from soup to nuts. If you are working against a maintenance window, this information is crucial. Assuming the testing was done on similarly configured hardware, the timing should be fairly accurate for the rollout in production. If the production environment takes significantly more or less time during any step, this is a good indicator of an area where the test environment is not sufficiently similar to the production environment.
- Make sure all steps to apply changes are well-documented. If they are not, reject the application of the change until they are. There should be *no* ambiguity as to what should be done, and no assumptions should be made, as you may have no control over who actually applies the changes. It may be that junior DBA hired yesterday.
- Always have contingency/fallback plans in place and ready to go. Risk mitigation is the key to change management. While most of the time whatever you are rolling out will be successful, you will encounter at some point a time when your update fails. You need to then find a way to quickly troubleshoot and complete what it is you are trying to achieve.

- Set a go/no go point, and stick to it. The go/no go point is when you meet with all parties involved beforehand to ensure that everything is lined up for the update to be a success, and if the implementation is taking too long and the maintenance window is drawing to a close, a call can be made to cancel the update and the rollback process can be started.
- Once the change is rolled out successfully, test. Even if the change deployed is a patch to Windows, which may have nothing to do with your application, you should have testers on hand (or be able to remotely log in, even if it is 2 a.m.) to ensure that what was rolled out did not affect the application. You do not want to roll out a hotfix on a Friday night, only to discover on Monday morning that no one can connect to the application.
- Do a postmortem after the update is applied even if it is successful. There are always lessons that can be learned and applied to future updates that may be similar, and the lessons, whether painful or positive, should be used as reference material for future work that may need to be done in the future. For example, if something happens that has a workaround, there is a chance someone may encounter that again. If you only have a postmortem and do not document the findings, it is a missed opportunity.

The Cost of Availability

On the surface, there seems to be only one cost to achieving availability: the financial cost of implementing the solution. That is certainly important and will partially drive the process to varying degrees; but there is another cost that must be taken into consideration. What does it mean if the business incurs a system-down event? This answer *should* directly influence the actual cost of the system and is discussed in detail in Chapter 2.

For example, assume that you work for a company that sends specialty automobile parts to dealerships all over the world. You are linked via a business-to-business (B2B) system where your customers order on a web page and can view your current inventory. You are located in Chicago, where your standard workday hours are not the same as those in Tokyo, Frankfurt, or Cairo because you are not in the same time zone. With this system, you are nearly able to process each request instantaneously, and since putting it in place, your profits have soared 300%, and most orders are seeing a 12-hour turnaround from order placement to fulfillment, when it used to take nearly three to five days. Financially, you are now making on average $100,000 per day in orders, which translates into just under $8,500 per hour. The company has blossomed from one that nets $500,000 per year and just covered its expenses, with a little room for growth, to one that is now seeing profits of $15,000,000 annually.

Based on pure numbers alone, for every hour of that B2B system being down, you are losing a serious amount of income that may go elsewhere and never come back since other companies carry similar merchandise. One of the basic rules of any business, whether it is consulting or selling auto parts, is that relationships are everything. Damage relationships and you'll spend more time, effort, and money replacing the customers you lost or trying to woo back the spurned.

Another example is a crucial hospital system dealing with patient care. The system provides essential information about patient history and drug allergies. If the system goes down and there is an emergency, the time lost could mean the difference between life and death. Seconds count.

A final example is one that everyone has most likely experienced: waiting to pay by credit card at a store. How many times have you been in a store, walked up to the sales clerk, and gone through the transaction, but when it comes to authorizing your payment, the system is down, busy, or unavailable? This gets magnified even more at the holiday shopping season when lines are long and additional time in line means the queue gets longer, customers become irate . . . the list goes on.

Defining Downtime

Downtime is much more than the failure of a database or SQL Server instance and the length of time it takes to come back up. Downtime includes the events that occur, the repercussions afterward, and the dependencies that go along with everything involved. SQL Server is usually only one component in the whole solution that needs to be accounted for. Downtime can be affected by numerous things, and you must plan for an appropriate budget for mitigation. This budget must include but is not limited to the following:

- Proper staffing
- Hardware acquisition
- Software licensing
- Support costs
- Yearly improvements (both software and hardware, including hardware refreshes, software upgrades, software license renewal, etc.)
- Disaster recovery

Downtime in isolation is only an amount of time. When talking about system and application availability, there are two main types of downtime that you need to define in terms of how your organization works: *planned downtime* and *unplanned downtime*. Beyond that, there is also the concept of *perceived unavailability*.

Planned downtime is the simplest to understand. It is when you have a known outage that is taken into account and hopefully well-communicated to all who need to know. For example, most environments have scheduled outages to perform maintenance on servers that will disrupt overall availability. Usually these outages happen during low-usage times, such as the early morning hours during the weekend, and can be short or long in nature depending on the work that needs to be done and/or the agreements that dictate how long the servers can be kept "offline," meaning even if the server or application is up, it may not be available to end users.

Unplanned downtime is exactly what I am trying to help you prevent in this book: when things go down without warning and you have no idea why. This is the worst-case scenario that no one wants but that happens to everyone at some point in their careers. I cannot state enough that there are no 100% guarantees against unplanned downtime, even if you follow all the advice and best practices in this book and seek out the guidance of others who know what they are doing. But you can certainly have the plans in place to mitigate it if it happens.

Perceived unavailability is arguably the trickiest to define and plan for. It may have elements with aspects of both planned and unplanned downtime, or it may have its own characteristics brought on by other factors. The easiest definition is when the end users can connect to the application or solution, which includes SQL Server as the back end, or directly to the database but do not get results in a time frame they expect. This may appear as perceived unavailability to the end user.

A good example of perceived unavailability is when an application is having performance issues, causing problems such as locking and blocking at the database level, or slower performance due to disk I/O being saturated from other requests to the database. Another good example is when SQL Server is up and running but the web server that processes the requests is down. To the DBA, there is no problem—SQL Server is up and running—but the users cannot access their data, so they are complaining to your help desk or management and all of a sudden the spotlight is on you.

Perceived unavailability can be prevented more often than not and is discussed further in this book as topics such as service level agreements (SLAs) and proactive monitoring are introduced. However, no matter what you do, if this type of situation occurs, you cannot help that the perception from end users will always be the same, whether it is a true outage, slow performance, or not even SQL Server's fault.

Nines

With both high availability and disaster recovery, you may have heard the term *nines* floating around. The concept is a simple one: a *nine* is the number of nines in a percentage that will represent the uptime of a system, network, solution, and so on. Based on that definition, 99.999% is *five nines of availability*, or the measure that the application is available 99.999% of the year. But how should a nine be interpreted in the real world, which is definitely not the absolute value of what 99.999% translates into in minutes?

Before you figure out what level of nines you need, you must determine whether your planned downtime counts toward your overall availability number. When defining availability, I always like to include both planned and unplanned downtime. Leaving out planned downtime gives you a skewed picture of your actual availability in the time frame you are measuring. For example, it is easy to say you achieved 99.999% uptime this year in a report to upper management. What you might have failed to mention is that your application and systems were only mission critical from 9 a.m. to 5 p.m. in your environment, and you had no outages *in that time frame between 9 and 5*.

Calculating the number of minutes allowed by each level of nines is straightforward: take the number of hours per year (365 days × 24 hours × 60 minutes, or 525,600 minutes per year), and then multiply your percentage desired. Simply put

$$\text{Minutes Allowed to Be Down} = 525{,}600 - (\text{percentage} \times 525{,}600)$$

Table 1-1 shows the amount of time associated with the most popular levels of nines.

Table 1-1. *Nines and the Downtime Associated with Them*

Nines Percentage	Number of Minutes of Downtime (Yearly)
90.0	52,560 (876 hours)
99.0	5,256 (87.6 hours)
99.9	525.6 (8.76 hours)
99.99	52.6
99.999	5.26

You may already be thinking to yourself that 5.26 minutes of downtime *per year* is impossible, and you would nearly be correct. It can be achieved, but the amount of effort and money you will spend to acquire, plan, deploy, manage, and maintain such a solution is exponential with each additional nine and may not be worth it in the end.

The reality is that while five nines is nice to have, most companies are able to perform the necessary business and IT activities (including maintenance) and achieve three to four nines. Even three to four nines can be difficult for some of the most seasoned IT organizations. Four nines, or 52.6 minutes, equates to just more than 4.38 minutes of downtime per month. Is your organization currently able to achieve that? Maybe, but three nines equates to approximately 43.8 minutes of downtime per month. That is arguably a more realistic number to strive for, especially if you have no experience with mission-critical systems to begin with.

Out of the gate, if you have a lack of experience and you call the Availability Fairy to come in and sprinkle fairy dust over your servers, I would be surprised if you got more than three nines. I may be kidding around a little here, but three nines is nothing to sneeze at. It is an achievement if you are coming from a place where you previously had hours of downtime. Bottom line: work up to more nines. Shooting for the moon is noble, but it could prove costly and fail miserably. Be realistic about what you can and cannot do. Do not be afraid to communicate that to upper management either, and make sure they are in the loop as to the limitations of what can and cannot be done since their planning and responses may depend on it.

Why did I bring up management? There is something to be said for the CEO/CFO/CIO/COO/CTO (who will be referred to in the rest of this book in examples or scenarios as the *CxO*) who may want or demand to have five nines but then does not commit to the expense of the human and actual cost associated with it. Management, for obvious reasons, might want to make magnanimous announcements such as "we will achieve five nines" and announce it to the world in some blustery press release if what you are doing is externally facing, but it is up to the groups who are tasked with implementing to push back where appropriate. If you do not push back and get everyone on the same page as to what can actually be done, there could be even worse consequences down the road (up to and including people being fired).

■ **Note** Never forget that you are only as good as your weakest link. Take SQL Server—when it is installed in a stable platform, you may be able to achieve three to four nines right there. But when is SQL Server the only component of a solution? Never. So if SQL Server is the back end of a BizTalk solution that involves the network, a storage area network (SAN), web servers, BizTalk application servers, and the rest of the ecosystem supporting it (including people and processes), how available is each one of those components? If your web servers are down and they are the front end to the users, is SQL Server really up? Technically it is, but it most likely is not servicing any transactions since nothing is coming in. This helps frame the content of Chapter 2, where you will see that the number of nines you need is driven by many factors, and you will learn how to define your availability criteria.

If you want to calculate your actual availability percentage, the calculation is the following:

Availability = (Total Units of Time – Downtime)/Total Units of Time

For example, there are 8,760 hours (365 days × 24 hours) in a calendar year. If your environment encounters 100 hours of downtime during the year (which is 8⅓ per month), this would be your calculation:

Availability = (8760 – 100)/8,760

The answer is .98858447, or 98.9% system uptime, which is nothing to sneeze at. To say your systems are only down 1.1% of your calendar year would be an achievement for some environments.

The Data Center Mentality

Whether you are a small company with one server or a large enterprise with thousands, the same guiding principles apply to making your systems available. Obviously the scale of execution and the inevitable budget considerations will most likely be different. The problem, however, is that no one has an unlimited budget. So what do you do?

To start forming your data center mentality, look at what has already been presented. Start with your people and your processes because technology alone will not solve things. No matter how automated you are, you still need human beings somewhere behind the scenes. Automation is going to be key in maintaining availability, but technology itself is not the panacea. High availability should never be a turnkey mentality (despite what some vendors may tell you when they are selling you a solution). The primary reason is that the worst form of downtime comes from problems that arise unexpectedly. These problems, by their very nature, are going to come from left field where nobody will have expected them. As a result, the automation is not going to have a custom-coded handling routine for them. However, the right people following a good and tested set of processes are the match for any problem.

The data center mentality when it comes to systems is a whole different mindset. How many development or test servers do you have in your environment that are down for one reason or another with no ETA, or are down intermittently because some developer made a change in code and then had to reboot it, which in turn kicked everyone else off? That just cannot be allowed to happen in a 24x7 production environment. People lose their jobs over these things, and I have seen it happen.

Availability is serious business that has equally serious consequences that must be in place. The old adage "if you can't stand the heat, get out of the kitchen" definitely applies here. Anyone who has been in the industry for any length of time has his favorite server-down story, which usually involves many long hours away from family, friends, and life in general. The story at the time was not funny, but looking back, it is amusing to think that it could have happened. In most cases, the signs of impending doom are obvious in hindsight and overlooked for one reason or another.

In many companies, and especially in ones with Windows-based applications, some systems start out as someone's desktop, but through some process that seems to happen naturally, that machine all of a sudden has some central role and becomes a production server without really having been properly planned to become one. That "server" winds up staying in Bob, Raheel, or Miko's cubicle and is vulnerable to everything from the cleaning crew unplugging it to use the vacuum cleaner to someone accidentally bumping into it and resetting it. This is absolutely a situation that does not lend itself to making that server a highly available system and preventing downtime.

The rest of this section will examine the things your company needs to take into consideration to ensure that your data centers and servers are availability and disaster ready.

Planning the Data Center

One of the most difficult things is actually planning your data center. A lot of the reason has to do with obsolescence: how long do you expect to use the data center? The amount of time is directly related to capacity planning. Miss your estimate and you may run out of capacity sooner, which brings its own host of problems. When talking about data center capacity, it is much more than just physical room for servers and racks; it is electric consumption, air conditioning, noise control, physical security, and so on.

The first thing you need to do when planning a data center, or reevaluating a current data center and its design, is to figure out capacity. How long do you plan on having this data center in operation? How much growth is the company experiencing, and does the lifespan of the data center meet the growth pattern? Oftentimes this is what gets people in trouble. For example, you should know the number of servers on average you are adding per month to the data center.

This example is the "simplistic" approach and must be done no matter what. A more detailed approach would examine your IT organization and branch out from there. For example, a retailer who exclusively sells online may have quite different needs than one who has a chain of "brick and mortar" stores (possibly in addition to an online presence).

Taking the next step, what is the data center going to house? Is it going to house critical infrastructure, or just servers? Some of both? The bottom line is that you need to know what you are designing for. Keep in mind that technology will change the landscape, but you need to change with it. A few years ago you may have bought 4U database servers that were racked, and now you are considering blade servers with just as much (or more) computing power that take up less physical space. However, you keep the same 10/100 network backbone and do not upgrade its capacity. Does that make sense? Maybe it does, maybe it does not, but it all needs to be evaluated.

Here are some decision points to consider:

Choose your data center location: This is multidimensional; there are geographic region considerations as well as local considerations. Unfortunately, as weather-related disasters can happen and will continue to happen, if the data center is located in a part of the world that is susceptible to potentially catastrophic natural disasters such as hurricanes and earthquakes, you must take this into consideration when planning, and take the appropriate measures. Whether that means reinforcing the building to ensure that the servers do not fall down during an earthquake, or somehow waterproofing the room, weather is not a server's best friend.

The other main aspect of location—where the server is located in a building—is not always straightforward. You must know the building's layout. For example, you wouldn't knowingly place your data center in a basement under the water pipes of the building or where there is not enough clearance for the racks. The basement seems like a good idea until the main water valve bursts, right? You must do your homework, as what may be an ideal location from a space perspective may not be right for other reasons.

Size your data center appropriately: Capacity management is more than sizing a server with processors, disk, and memory. It applies to building out your data center, too. Much like when you buy your hardware, you have a certain life expectancy of the solution. Do you want to keep the data center for five years? Ten? Indefinitely? If the answer is indefinitely and you are a rapidly growing company, you may hit your capacity sooner rather than later without proper planning.

Install climate control: Whether you live in a tropical climate or in Alaska, your data center must account for it. Servers generate heat, and lots of it. The more servers, the more heat is generated. This should be taken into account when sizing your data center and figuring out how long it will be in service. You should not have to replace your climate control system midway through the life of the data center. Servers do not perform well in heat, and their proverbial lives are reduced when there is not enough air cooling the data center. Some systems may actually perform faster when cooler. You also need to ensure that the design of the data center allows for proper air flow above and below the systems (including in the racks themselves), and the air flow should be even in all parts of the data center. You do not want any one spot being too hot or too cold.

Rack your systems: Putting your servers in racks saves space, but make sure that the racks you buy can actually fit in the data center you are going to use. Make sure that there is enough room above and below for air flow, and that there is sufficient room for cabling; so measure your dimensions of the doors and room prior to purchase.

Install fire suppression systems: Fire suppression is necessary, but make sure you install a system that will not damage your servers or endanger human lives. There should be visual cues (such as a blinking light) as well as audible clues (such as loud alarms) in the data center to ensure that any workers in the data center can exit the room safely. The data center is usually a loud place, so without those measures, there could be the potential loss of human life.

Document the physical server installations: This may seem painfully obvious, but there is nothing worse in diagnosing a hardware problem than trying to figure out not only which rack and space the server is in, but what hardware is configured in it, and where the connections such as fiber cables to a shared storage array are plugged in.

All of the previous points will influence how your data center is built and constructed and can ultimately be easily quantified in terms of money; but there is more to building a data center than the physical aspects. Spending millions of dollars only to skimp on some other aspect of your data center deployment is akin to shooting yourself in the foot. The points listed are only part of what needs to be taken into account in the design of the data center itself.

Securing the Data Center

If there is any one thing that needs attention from the beginning of the design of any application, system, or location, it is security. Security has many levels: physical security of the facility and access to the servers themselves, right down to how applications and clients interact with the solution. Certainly external threats such as denial of service attacks against web servers need to be defended against, but arguably the biggest threats to security come from the inside. Corporate espionage or disgruntled employees (or former employees who somehow still have access) are much more damaging, and by the time the damage is discovered, it is most likely too late. There are certain precautions that can help.

Keep Systems in the Data Center

If a system is relied upon by more than one person, you should most likely move it into the data center. How many applications start out as non–mission critical and are located on someone's workstation or a development/test server, and then for one reason or another it becomes something the business cannot live without? That is a good thing, but the location of the server is still in someone's office where it can easily be tampered with. As crazy as this may sound, this scenario still happens every day. I hear the stories time and time again. Do you want the cleaning staff to accidentally unplug the server so they can vacuum? That would cause an availability outage that not only could be prevented but should never happen in the first place. Even worse is when the DBAs, the help desk, or other administrators get a call saying the system is down when they never knew it existed in the first place. That is never a situation anyone wants to find themselves in, as the blame game will start to be played full force.

Control Access

Make sure access to the data center and server room is tightly controlled. Formalize plans around who will have access to the data center, what they can touch, when they can touch it, and how they will be able to get in and out with a trace of who was actually in the data center. Even if it is as simple as a sign-in and sign-out list on a clipboard, it is better than nothing. Ideally, there would be electronic pass cards or other advanced technologies such as biometrics that would more accurately track the comings and goings in the server room.

You should maintain a backup key to the door (i.e., a way to get in) if the electronic security measures are not working. Achieving this level of security can get difficult if third-party vendors or repair people need to come in and perform some service or install new equipment, but there should be formalized processes on how those people are allowed into the actual server room. In the case of third-party vendors, most companies have some sort of an escort policy where the visitor must be with an employee at all times.

Install Surveillance Equipment

Along with processes around who gets in and out of the data center and how that happens, you should have a visual record of it. Did a server go missing last weekend? Who really was in the data center? Electronic measures such as pass cards unfortunately can be circumvented or impersonated if someone is really good at doing such devious things. If you have still pictures or some form of video, it is easier to actually *see* who was there. Implement a formalized retention policy around how long surveillance pictures or video will be retained.

Lock and Secure Server Racks

Do not let everyone and their uncle have keys to all racks and server locks. Locks may be low-tech, but they are a great deterrent to thieves and others who have malicious intent or might sell the equipment for a quick buck. Only the owners of the system as well as those responsible for maintenance should have the keys and the ability to remove the systems from racks and be able to access sensitive parts, such as the fiber cables leading from your SQL Server to the SAN.

Enable Remote Administration

Back in the stone age of server administration, more often than not, all work done on the server, either at an operating system level or physically to the server itself, was done in the data center. Times have changed, and third-party products such as Citrix or the built-in Terminal Services feature of Windows allow administrators to access the servers and do their work without physically having to be in the data center. With SQL Server 2005, you have tools such as Management Studio that allow you to do most DBA-related tasks remotely. The more remote management solutions can be implemented, the less risk there is to the physical (not logical) environment. Securing the *logical environment*, meaning the operating system and the applications running on top of it, is a whole different task.

Secure Your Data and Backups

When it comes down to it, no matter what the application is on top, you are accessing data that has some corporate value. Secure it properly by restricting access and adding encryption if necessary. The same goes for your SQL Server database backups. Once you make the backup, make sure it is placed in a secure area (who is preventing someone from taking it and restoring it elsewhere?) and is encrypted.

Having said that, a major issue when it comes to hiring the right staff for managing your database environment that I hear all the time is summed up in the following question: how can you prevent DBAs and/or administrators from viewing data that may be sensitive or confidential? The answer is simple: do not hire people you do not trust. While security will be discussed all throughout this book and data security is a subset of that discussion, this requirement is much more basic than that. Only hire people that you trust will not do anything malicious with any data they may have access to. This type of security has nothing to do with accessing servers and data with accounts configured with the least amount of privileges possible. Technology solutions can help in terms of securing data, but you can never guarantee that it will do the entire job. Sometimes DBAs or administrators have to view data to do their jobs.

To match any technology solution for minimizing access to sensitive data, there should be corporate policy measures to bolster your security. For example, there could be a policy that lists penalties, such as payment of any financial damages if someone leaks information, loss of their job, or other severe penalties that match the damage incurred. Assuming the penalties are stiff, they should serve as a deterrent to anyone who wants to try to do something harmful.

What makes all of this harder is that some IT organizations are outsourcing operations to contractors, third-party hosting providers, or remote offshore organizations. The same rule applies: hire well and you will have no problems. Just because you may not see these people and they are not employees who are "full" members of your organization does not absolve you from having control over the hiring process, who touches your environment, and the penalties associated with anything that happens as a result. Any outsourced employees should be held to the same standard as full-fledged, salaried employees.

Audit Logs for Security

Most applications such as SQL Server and operating systems such as Windows have ways of auditing security. While a user *could* be impersonated, the only surefire way to track something is to be vigilant and see who is on the server and what they are doing. Auditing within SQL Server itself can add overhead, but when someone does something malicious, you will want to know what was done, who it was done by, and what the extent of the damage is. Along with auditing logs, you should arguably have some way to trace a connection back to a particular workstation/NIC/IP address.

■ **Caution** Applications should *never* be coded to have system administrator (SA) access within SQL Server because someone can come in through your firewall who knows the user account with SA privileges and its password, and then proceed to wreak havoc on your SQL Server instance. Do not find this out after it is too late. Ask questions of your developers early and force them to code to a least-privileged user.

Devise and Enforce Corporate Password Policies

Never use a master password that can access all systems. This may seem obvious, and a data center staffer managing a lot of servers may hate me for saying this, but if this password gets out, someone may get a backdoor into a system. Most software such as SQL Server requires domain-level service accounts, and changing these accounts is a huge pain and may even cause downtime. But what is the cost if someone who should not accesses sensitive data? A few seconds of downtime to cycle the SQL Server service is generally a small price to pay. Another consequence of using a single service account is that if all of your SQL Servers use the sqladmin account in a particular domain, you will have one massive outage to change the password for each instance. That has to factor into your overall availability picture.

■ **Caution** Depending on how your password security is devised, you may affect your availability. For example, Windows has the ability to ensure password expiration after a certain number of days. Resetting this password may affect the availability of applications such as SQL Server that may use domain-based Active Directory accounts that need to be reset. These types of things must be taken into account when solutions are devised and implemented.

Isolate Users and Administrators

Some environments require that a master administrator account is created and has access to all servers. Other than this possible user, all other users, logins, and administrators should strive to be unique to isolate any potential problems.

Power

Without power, systems cannot run. It is as simple as that. Therefore, you have to make sure that there is no interruption in power to your servers. The following are some things to help you mitigate power issues in your data centers:

Ensure you are deploying the right power for the equipment: There is nothing funny about putting a 110 volt plug into a 220 volt power source and vice versa. Standardize on how things will be plugged in, deploy that, and make sure every acquisition adheres to that standard.

Make sure all electrical lines are grounded and conditioned: Grounding is essential for computers. Some older buildings may not have grounded wiring and must be rewired to support grounding. To ensure that all things that are in the data center are powered evenly, and that there will be no sudden power spikes that could fry internal components of your servers, use power conditioners to provide even power to all of your servers. Conditioning also prevents undervolting, which could damage the system by not providing enough power.

Deploy backup power supplies: Each server and separate disk storage should have some sort of uninterruptible power supply (UPS) attached to it to allow it at a bare minimum to be shut down gracefully in the event of a data center power failure. Some data centers will implement central UPS solutions that will work with all equipment used in the data center.

Plan for an extended power outage: A UPS will only protect the device it is connected to and will provide a short amount of emergency power to it, not the whole data center. Depending on how much power the UPS provides and the time it takes to shut down the servers, the UPS may or may not provide enough protection. During the concerns over Y2K, many companies bought large diesel-powered generators to power their data centers in the event that the power went off at 12:01. You should consider deploying a similar solution to power your business if it needs such availability requirements.

Buy a battery backup solution: Battery backup is most common with storage devices such as SANs. With SQL Server, this becomes especially important since once the data goes through and the hardware handshakes with SQL Server saying in essence "I've got your data," there is a period of time (even if it is milliseconds) before the cache is flushed and written to the physical disk. If the power goes out and you have no battery backup, you may potentially lose data or corrupt disks.

Monitor equipment batteries: Remember to check the batteries in your controllers and other components that contain them, and make sure that they are replaced before they fail. If a battery dies on a key component, you may cause worse problems than you can even imagine.

Cabling, Networking, and Communications

One of the major components of your data center is the miles and miles of cabling that connect everything together. Properly cabling a data center to support your network and communications infrastructure can pose significant challenges since you will have power cables, network cables, fiber cables, and more. These tips will assist you in the process:

Keep all cabling neat: There is nothing worse than trying to troubleshoot a hardware problem only to discover when looking at the back of a rack of servers a rat's nest of cables. You cannot trace a cable from its source and destination that way. Do due diligence and cable your systems, networks, and storage properly. You may be under the gun to get things done quickly, but that is no excuse for sloppy work. In the event of a failure due to a cable problem, it is much easier to pull and replace when it is not tangled up with 1,000 others. Time spent doing things right at this point pays huge dividends later.

Properly label all cables: Along with neatly putting cables into components, buy a labeling machine and label both ends of the cable. If you are going to spend thousands, or even millions, on your data center, investing $50 on a labeler is one of the biggest bargains in your budget and can even be used for other purposes such as labeling your servers. You may also

want to consider using color coding for each cable that corresponds to its purpose. For example, all network cables will be blue, all crossover cables yellow, and so on. If a single cable needs to be replaced for some reason or is knocked loose, one of both of these methods will make it easy to identify them.

Install phone lines in the data center itself: Whether it is your employees needing to talk to someone internally, or a third-party vendor needing to call some people while he or she is in the data center, you need a phone installed. Although it is arguably outdated, you may also want to consider standard analog phone lines in addition to a digital phone system in case the digital phone system fails; this would enable modems in servers or laptops to dial out of the data center. For the über-paranoid, install a pay phone or locate your data center near one. Pay phones operate on completely separate circuits that almost always stay up during emergencies.

Have backup communications systems in place: In the event of a power outage or another type of emergency where phone systems may be down, make sure you can communicate. Whether you use walkie-talkies, pagers, or some other device that meets your needs, do not assume that your primary communication system will always be working. For example, if you only have one phone in the data center and there is no cellular phone reception, you may have an inability to solve your crisis and keep management informed. If you have redundant communications, your coworker can be talking on one line on a phone to tech support while you are on another line talking to management.

Employ redundant networks: In the event of a main network failure, you should have the ability to switch to an alternate source to have traffic flow in and out with little to no downtime. If your main network fails, every single server in your data center, and subsequently all of your company's productivity from the top on down to end users, will be halted. Yes, this is expensive, but a network blip can also cause more havoc than just reducing productivity. It may introduce instability into such things as applications (depending on how they handle such failures), and could cause a whole chain of events to happen. This scenario, however, may not take care of a wide area network (WAN), which involves multiple sites that are located across the globe. A redundant network will only protect the network in the location it is deployed.

Include the network in capacity planning: Imagine this scenario: your data center has plenty of free rack space. You buy a new set of servers for your new SharePoint installation, get them racked, and shockingly discover you are out of network ports or the necessary amps to run them. Need I say more? The best test is the "9 a.m. test" where you turn everything on at once and see what happens. You can oversaturate everything in your data center very quickly.

Outsourcing Server Hosting and Maintenance

Given the costs of implementing, maintaining, and staffing a data center, companies are increasingly hosting their servers in a third-party owned data center that may or may not be near to where the company is actually located. If this situation is one you are currently in, or one you are considering, the following points should help you in implementing a data center, whether you are just leasing space with your engineers onsite or you are fully outsourcing everything. Pick your hosting company wisely—your availability will depend on it.

Ensure that your employees can access your servers: What access will the hosting company actually give you if your administrators show up onsite? Is there a limit to how many of your employees can actually be in the data center at one time? Does the hosting company restrict access to its facility at certain hours, or do you have 24x7 access? You need to look beyond the monetary cost you may save on paper to see if any of the answers to these questions will actually increase your downtime in a server-down situation. Also, know how much it will cost to go to the data center. Is it a plane ride? Do you need hotels and rental cars?

Know how long it takes to get to the third-party facility: If for cost reasons your third-party data center is not close to where your company is located, know how long in both worst-case and best-case scenarios it takes to actually get there. In a normal situation where you are going to just do some scheduled maintenance, time may not be as much of the essence, but in a server-down situation, seconds, minutes, and hours will count, depending on what SLAs you have in place. Third-party data centers will most likely have their own staff and can certainly act as your "eyes and ears," but you should always know what it takes when your staff needs to be onsite.

Qualify the hosting company: If the hosting company will also be responsible for multiple tasks, including administering your systems, what support agreements do they have in place with their vendors and how will they affect you? For example, if they have a contract with a network provider that states the network provider will have a repair person onsite within 24 hours when you need to be up in minutes, is that acceptable? Are their own policies and procedures in line with your overall availability goals, and will they happily accept whatever corporate policies, procedures, and standards you want them adhere to, or will they reject them?

■ **Tip** A good rule of thumb is that the hosting company should be capable of and already delivering to existing customers one more nine than necessary. This will give you a sense of confidence that it can meet your needs. Ask for references.

Consider turnaround time: If you decide to go with a third party, most likely you are bound to its infrastructure and rules for fixing problems, especially if it is the primary caregiver for your equipment. If the engineers cannot meet your SLAs and are not up to what you would consider to be a high enough level of expertise, those are warning flags. Make sure your agreements specify how long it will take if you call and issue a trouble ticket, that you have direct access to a qualified engineer, and that you don't have to go through a tiered and gated system. If you're talking to the wrong person, it will only delay getting things fixed in a server-down situation.

Put procedures in place to monitor third-party work: How do you know whether your requests, for example, to apply a SQL Server service pack, are getting done? Out of sight should not mean out of mind. You are entrusting your servers to people you may not have daily interaction with, so there needs to be a formalized process in place to foster communication around these types of activities.

Ensure all equipment is secure: If your servers are in a hosting company, are you the only customer? If not, where are your servers in relation to other companies? And if there are other companies in the data center, what policies or other measures are in place to prevent them from touching your servers?

Technology

Availability is straightforward from a pure technology perspective and can be summed up in one word: redundancy. For example, if you have one server as your primary database server, buy another and configure it as a "backup" (so to speak). But that is not all you need to do. Providing the right redundancy is far more complex than just buying another server and putting it on the network. Redundancy, to some degree, *is* the key to almost everything that will be presented in this book; and *how much* redundancy you need is dictated by how you will ultimately define your availability criteria.

Summary

Availability is not something you will achieve overnight and is not something that can be achieved by throwing money and technology at it. Downtime, especially unplanned downtime, is a matter of when, not if, you will encounter it, so investing the proper resources to mitigate that risk will be necessary. Do not think of the upfront cost, because over time it is amortized. Think of the peace of mind you will have knowing you will be able to handle most disasters that come your way. Chapter 2 will walk you through other aspects of preparing for availability, so come along on the rest of the journey to see how to make SQL Server 2005 highly available.