International Technical Support Organization

Best Practices and Tools for Creating WebSphere Commerce Sites

February 2005
Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Second Edition (February 2005)

This edition applies to Version 5, Release 6, Modification 2 of WebSphere Commerce.
# Contents

**Notices** ................................................................. ix
**Trademarks** ............................................................ x

**Preface** .................................................................. xi
The team that wrote this Redpaper ........................................... xi
Become a published author ....................................................... xii
Comments welcome ............................................................ xii

**Summary of changes** ................................................... xv
February 2005, Second Edition ................................................ xv

## Part 1. Systematic Project Development Method ............................................. 1

### Chapter 1. Introduction ......................................................... 3
1.1 Further information ......................................................... 10

### Chapter 2. Method for project development ........................................ 13
2.1 Systematic development method ............................................ 14
2.2 Definitions .................................................................. 14
  2.2.1 Work product ......................................................... 15
  2.2.2 Deliverable ............................................................ 15
  2.2.3 Phase ................................................................. 15
  2.2.4 Activity .............................................................. 16
  2.2.5 Task ................................................................. 16
  2.2.6 Work unit ........................................................... 17
  2.2.7 Work breakdown structure ....................................... 17
  2.2.8 Strategy ............................................................ 17
  2.2.9 Customer .......................................................... 18
  2.2.10 Customer IT team .................................................. 18
  2.2.11 Project team ........................................................ 18
  2.2.12 Project repository .................................................. 19
  2.3 Project life cycle .......................................................... 19
    2.3.1 Core development phases ..................................... 21
  2.4 Using the method ........................................................ 23
    2.4.1 Customizing and adopting the method ....................... 23
    2.4.2 Project roles and skills requirements ......................... 23
    2.4.3 Structuring information .......................................... 24
    2.4.4 Case studies ...................................................... 24
  2.5 Summary ............................................................... 24
### Chapter 3. Design phase

- **3.1 Purpose of the design phase** ............................................. 28
- **3.2 The design phase** ......................................................... 29
  - **3.2.1 Creating the site design** ........................................... 30
- **3.3 Solution outline activity** .............................................. 34
  - **3.3.1 Gathering the functional requirements** ......................... 36
  - **3.3.2 The solution outline workshop** .................................. 39
  - **3.3.3 The solution outline document** .................................. 41
  - **3.3.4 Analyzing current site assets** ................................... 42
  - **3.3.5 Design-based sizing** ............................................. 42
- **3.4 Macro design activity** .................................................. 43
  - **3.4.1 Methods of approaching design activities** ..................... 43
  - **3.4.2 The data model** .................................................... 46
  - **3.4.3 Creating interaction diagrams and class diagrams** ........ 49
  - **3.4.4 JSPs templates** ................................................... 51
  - **3.4.5 Creating a WebSphere Commerce site database** ............. 51
- **3.5 Micro design activity** ................................................. 54
- **3.6 Design review** .......................................................... 56
  - **3.6.1 Data analysis review** ............................................ 57
- **3.7 Summary** ............................................................... 58

### Chapter 4. Build phase

- **4.1 The project build phase** ............................................. 60
- **4.2 Build and team work assignment** .................................... 61
  - **4.2.1 Assigning team members to tasks** ............................... 61
  - **4.2.2 Tracking progress** ............................................... 63
  - **4.2.3 Regular builds and systematic integration testing** .......... 64
- **4.3 Integrated development and test infrastructure** ............... 65
  - **4.3.1 The team development environment** ............................. 67
  - **4.3.2 The shared test environment** ................................... 72
  - **4.3.3 Configuring a development process** ............................ 72
  - **4.3.4 Configuring a simulated production environment** .......... 73
  - **4.3.5 Source code management** ....................................... 74
  - **4.3.6 The build process** ............................................... 75
  - **4.3.7 Build automation** ............................................... 77
  - **4.3.8 Build failure** .................................................... 78
- **4.4 The problem determination process** ................................ 79
  - **4.4.1 Component installation and configuration** ..................... 80
  - **4.4.2 Build deployment** ............................................... 80
  - **4.4.3 Runtime problems** ............................................. 81
  - **4.4.4 Application or customization defects** .......................... 81
  - **4.4.5 Performance problems** ......................................... 81
- **4.5 Subsystem integration test activity** ................................ 82
4.6 Handling errors .................................................. 83
   4.6.1 Handling each aspect in a real project ................. 84
4.7 Summary .......................................................... 86

Chapter 5. Site test phase ........................................... 89
5.1 Site testing activities ......................................... 90
   5.1.1 Creating the site testing infrastructure and processes 91
   5.1.2 Executing test cases ........................................ 94
5.2 Functional Verification Test (FVT) .............................. 96
   5.2.1 Team organization during testing .......................... 97
5.3 System Integration Test (SIT) ................................ 98
5.4 System Verification Test (SVT) ................................ 101
5.5 User Acceptance Test (UAT) .................................. 102
5.6 Summary .......................................................... 104

Chapter 6. Launch phase .............................................. 105
6.1 Site deployment and launch activities ......................... 106
6.2 Summary .......................................................... 110

Part 2. Development and Migration Best Practices .................. 113

Chapter 7. Development and migration tools ......................... 115
7.1 Development tools ............................................... 116
   7.1.1 Lightweight WebSphere Commerce test environment 116
   7.1.2 Full WebSphere Commerce test environment ........... 117
   7.1.3 Comparison between the test environments ............. 117
   7.1.4 Command line utilities ..................................... 118
7.2 Migration tools .................................................. 135
   7.2.1 Database migration (migratedb) ......................... 135
   7.2.2 Encrypted data migration (migrateEncryptedInfo) .... 137
   7.2.3 Instance migration (wcim) ................................. 138
7.3 Additional tools ............................................... 140
   7.3.1 Lightweight Object Query System ....................... 141
   7.3.2 Auto-deploy tool ............................................ 142
   7.3.3 CommerceQA test tool ..................................... 142

Chapter 8. Developing new sites .................................... 143
8.1 Basing development on a starter store ....................... 144
8.2 Use separate database server .................................. 145
8.3 EJB and data bean best practices ............................... 146
   8.3.1 Customizing WebSphere Commerce tables .............. 147
   8.3.2 Customizing WebSphere Commerce EJ Bs ............... 148
   8.3.3 SQL security considerations .............................. 148
   8.3.4 Use well-formed ejbCreate constructors ................ 150
8.3.5 Lazy fetch data retrieval ........................................ 150
8.3.6 Handling database vendor specific code ...................... 151
8.3.7 Using session beans ............................................ 152
8.3.8 Managing concurrent updates. ................................ 155
8.3.9 Flush remote method .......................................... 156
8.3.10 Generating primary keys ...................................... 157
8.4 Best practices for JSP programming. ............................ 158
  8.4.1 Use JSTL in place of Java code ............................... 159
  8.4.2 Use WebSphere Commerce activation tags ................ 161
  8.4.3 Use WebSphere Commerce maps to get request parameters 163
  8.4.4 Use StoreErrorDataBean for error handling .............. 164
  8.4.5 Use an appropriate includes mechanism .................. 165
  8.4.6 Use the JSPF file extension for JSP segments ............. 166
  8.4.7 Use the escapeXml attribute to preserve HTML formatting 167
  8.4.8 Ensure XHTML compliance ................................ 167
  8.4.9 Use CSS to separate layout from content .................. 168
  8.4.10 Add id attributes to links and form elements ........... 169
  8.4.11 Invoking controller commands from within a JSP page .... 169
  8.4.12 Use the post method for form submission ................. 170
8.5 Command best practices ........................................... 170
  8.5.1 New command naming and hierarchy ....................... 171
  8.5.2 Using constants to hold class and method name .......... 172
  8.5.3 Command error handling .................................. 174
  8.5.4 Tracing .................................................... 178
8.6 General security considerations ................................ 179
  8.6.1 Access control ............................................. 180
  8.6.2 Encrypt sensitive information .............................. 180
  8.6.3 Protect sensitive configuration files ............... 181
  8.6.4 Protect sensitive parameters. ............................. 182
  8.6.5 Avoid default credentials and port numbers ............. 183

Chapter 9. Migrating existing sites ................................. 185
9.1 Why migrate? ...................................................... 186
  9.1.1 Business benefits of new products ....................... 186
  9.1.2 Business benefits of new capabilities .................... 186
  9.1.3 Prepare for end-of-service of your product ............. 189
9.2 Migration strategy considerations ............................. 190
  9.2.1 Getting help ............................................... 190
  9.2.2 Migrating the different environments .................... 191
  9.2.3 Handling complexity ...................................... 192
  9.2.4 Component view .......................................... 193
  9.2.5 Minimizing customizations ............................... 194
  9.2.6 Performance optimization with dynamic caching ........ 194
9.2.7 Scalability ................................................................. 194
9.2.8 Database cleanup ............................................. 194
9.2.9 Basing development on starter stores. .................. 195
9.2.10 Store archive (SAR) migration ............................ 195
9.2.11 Delta migration ..................................................... 196
9.2.12 How custom code impacts the process ............... 197
9.2.13 How database customizations impacts the process . 197
9.2.14 Migrating your WebSphere Commerce instance or recreating it . 197
9.3 Migration approach considerations .......................... 198
  9.3.1 Switchover migration ......................................... 199
  9.3.2 Coexistence migration ......................................... 201
  9.3.3 In-place migration ................................................ 202
9.4 Migration planning .................................................. 203
  9.4.1 Planning activities ............................................. 205
  9.4.2 Gathering migration requirements ....................... 206
  9.4.3 Planning the migration activities ......................... 207
  9.4.4 Preparing for migration ..................................... 209
  9.4.5 Migrating the site ............................................. 209
  9.4.6 Testing the migrated site ................................... 210
  9.4.7 Going live ...................................................... 210
9.5 Planning and sizing templates ................................. 211

Chapter 10. Integration with existing systems ................. 215
10.1 WebSphere Commerce integration capabilities .......... 216
  10.1.1 Messaging architecture .................................... 217
  10.1.2 Predefined messages ...................................... 218
10.2 Web services ........................................................ 219
  10.2.1 Overview ...................................................... 219
  10.2.2 Why use Web services? .................................... 219
  10.2.3 Web services architecture ................................. 220

Part 3. Appendixes ...................................................... 223

Appendix A. Project team creation and common roles ......... 225
Creating the project team ........................................... 226
  The core competency strategy ................................... 227
  The holistic strategy .............................................. 228
Guidance on selecting the strategy ............................. 229
Project roles ............................................................ 230

Appendix B. Analysis, site maintenance and enhancement phases 233
Analysis phase .......................................................... 234
  Analysis phase sizing ............................................. 235
Site maintenance and enhancement phase ..................... 236
Appendix C. Additional methods .................................................. 239
Deciding the method and terminology ........................................... 240
IBM methods ................................................................................. 240
The Rational Unified Process (RUP) ................................................ 240
Summary ....................................................................................... 243

Related publications ........................................................................ 245
IBM Redbooks .............................................................................. 245
Other publications .......................................................................... 245
Online resources ............................................................................ 245
How to get IBM Redbooks ........................................................... 246
Help from IBM ............................................................................... 247
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.
Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- Approach®
- Cloudscape™
- DB2 Universal Database™
- DB2®
- developerWorks®
- Domino®
- eServer®
- eServer™
- ibm.com®
- IBM®
- iSeries™
- Lotus Notes®
- Lotus®
- Notes®
- Perform™
- Rational Unified Process®
- Rational®
- Redbooks™
- Redbooks (logo)™
- RUP®
- Sametime®
- Tivoli®
- VisualAge®
- WebSphere®
- zSeries®

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.
Preface

Gone are the days when creating e-commerce sites was considered a fine art requiring highly specialized skills. E-commerce development has become a rather mature field since first introduced in the mid- to late-1990s. Experience proves that using a systematic methodology to develop software applications for e-commerce systems greatly reduces project risk and improves proper alignment of software functionality with customer business requirements, two key influencers of project profitability and customer satisfaction.

This IBM® Redpaper introduces a systematic project development method that was specifically adapted for the creation of e-commerce sites using the IBM WebSphere Commerce product family.

This Redpaper details two WebSphere Commerce development scenarios:
1. Creating a new e-commerce site.
2. Migrating to WebSphere Commerce V5.6.

This Redpaper covers development and migration tools and also provides best practices for both scenarios.

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.

Hernan Cunico is a Consulting IT Specialist Certified and WebSphere software specialist at the International Technical Support Organization, Raleigh Center. He writes extensively and teaches IBM classes on WebSphere Commerce and WebSphere Application Server. Hernan has nine years of experience in the Information Technology and e-business consulting areas. His areas of expertise also include networking, Internet security, e-business and e-commerce solutions architecture.

Nicolai Nielsen is an Advisory IT Specialist with IBM Global Services, Denmark. He has nine years of experience in the field of consulting, application development, and systems integration. Nicolai holds a degree in engineering from the Technical University of Denmark. He has written extensively on
WebSphere Commerce and has worked on several WebSphere Commerce projects over the last three years.

**Sanjeev Sharma** is a team lead of WebSphere Commerce Solutions team in IBM Canada’s software development lab in Toronto. He has 5 years of experience in WebSphere Commerce and database administration fields. He holds a Computer Engineering degree from McGill University in Canada. His areas of expertise include solution design, installation, integration and testing. He has written extensively on installation and integration test methodologies as well as WebSphere Commerce migration.

Thanks to the following people for their contributions to this project:

John Ganci  
International Technical Support Organization, Raleigh Center

---

**Become a published author**

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

```
ibm.com/redbooks/residencies.html
```

---

**Comments welcome**

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this Redpaper or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  
  ```
  ibm.com/redbooks
  ```

- Send your comments in an email to:
  
  ```
  redbank@us.ibm.com
  ```
Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HZ8  Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195
Summary of changes

This section describes the technical changes made in this edition of the paper and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for Best Practices and Tools for Creating WebSphere Commerce Sites
as created or updated on January 21, 2005.

February 2005, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information
- Best practices for development.
- Best practices for migration.
- Development and migration tools.
- Integration with existing systems.

Changed information
- Restructured content.
- Revised project development method.
Part 1

Systematic Project Development Method
Chapter 1. Introduction

Using a systematic method to develop software applications for e-commerce systems greatly reduces project risk and improves alignment of software functionality with a customer's business requirements, which are two key influencers of project profitability and customer satisfaction. This Redpaper introduces a project development method that was specifically adapted for the creation of e-commerce sites using the WebSphere Commerce family of products. Not only does it support new implementations of the technology, but the method can also be used to migrate current assets and data to WebSphere Commerce V5.6. In addition, the method may be combined with tools to streamline the entire WebSphere Commerce migration process.

A typical project development method consists of six phases. Within this paper, we focus on the following four core phases for commerce site development:

- Design
- Build
- Site testing
- Launch

A diagrammatic overview of the project development method is shown in Figure 1-1.
Figure 1-1  Overview of project development method
During the analysis phase, the customer must assess their business requirements objectively, and then plan their e-commerce site carefully, as results derived from these activities will set the direction and help define the duration of key project activities. It is critical for the customer to define and document all their business requirements adequately prior to the completion of this phase; the results produced by these activities will be the seed for the site design work.

The design phase marks the beginning of a WebSphere Commerce project, from the perspective of a services engagement, and during this phase, project teams work to define and document the site's design based on the requirements that result from the analysis phase. The design phase can be decomposed into the following three distinct activities:

- Solution outline
- Macro design
- Micro design

During the solution outline activity, the project team identifies all relevant functional requirements for the site from the predetermined business requirements. All functional requirements should be recorded in a solution outline document that would consist of the following components:

- Project team skills plan.
- High-level project plan.
- Revised sizing estimate (that is, work effort) required to develop the site.

A concluding task in the creation of the solution outline document is the solution outline workshop. The solution outline workshop allows the project team to work with the customer team, reviewing the project's functional requirements and plans. Where necessary, each may be adjusted to gain customer approval in order for the design work to start properly.

Prior to working on the design phase, the project team itself must be organized by the project manager or lead. The project team plays a significant role during the design phase in aiding the customer to define and document site requirements and project plans.

This team must consist of dedicated, suitably skilled individuals who can be relied upon to complete all required tasks related to the design and implementation of a target site. A project manager is likely to assess the following project factors when identifying team resource requirements and considering how best to structure a project team:

- Skill and experience requirements.
- Project objectives.
- Project timeline.
Customer requirements.
Geographic considerations.
Direct customer involvement in project tasks.
Training requirements.

A thorough understanding of these factors will enable the project manager to pursue an appropriate strategy for organizing and structuring the team accordingly. It will also enable them to staff the team with appropriate skills to match the project and customer objectives. Two proven team assignment strategies are as follows:

Core competency strategy.
Holistic strategy.

The core competency strategy advocates using expert resources to staff and complete project tasks based on skill. This strategy fosters specialization, because team members are able to hone and refine their expertise in given areas. Generally, when team members complete a project that employed this strategy, they are capable of undertaking more challenging design and development responsibilities in the future.

The holistic strategy, on the other hand, stresses using resources with strong fundamental e-commerce design and development skills to make each fully capable of performing all project tasks. One result of this strategy is a workforce that is versatile and adaptable at performing diverse work assignments. In addition, it enables the project manager to reposition resources as needed during the life of the project to aid the team in maintaining the integrity of the project schedule.

The project manager may also use an appropriate combination of the above strategies to satisfy particular project demands. Again, it is critical that the project manager fully assess the nature and scope of the project to determine how best to structure a project team.

It is during the design phase that the project team strives to create and document the project's functional requirements and requisite project and skill plans.

The project team is likely to employ one of the following strategies to complete the remaining activities within the design phase:

Iterative and incremental design strategy.
Combination design strategy.

The iterative and incremental design strategy advocates working through each of design's activities in sequence so that details captured during the earlier solution outline activity can be used and further extended by the subsequent activities. In other words, the site design gets built and refined over time as the team works
through each activity. This strategy work best on projects involving complex business logic.

The combination design strategy, on the other hand, enables the project team to combine the activities of related activities into one phase. For example, solution outline may be combined with macro design, or macro design and micro design may be combined. This strategy is suited for projects with simpler designs, because it allows the project teams to complete related design activities in parallel to reduce the length of time need to complete the project.

The main focus of macro design is the creation of a WebSphere Commerce design that incorporates the requirements that were approved during the solution outline. The project team may pursue one or a combination of the following design methods:

- System level (for example, WebSphere Commerce interacting with external systems).
- Subsystem level (for example, interactions among WebSphere Commerce subsystems).
- Use case level (for example, site flows as identified in use cases).
- Component level (for example, interactions among JSPs and EJBs).

These factors enable the project team to understand and then design a site given the approved requirements. Very detailed examples and project recommendations are provided throughout this Redpaper.

Once the macro design requirements have been defined and documented in the macro design document, the project team may then proceed with the micro design activity. The main objective of this third iteration of the design phase is for the project team to identify, and then document, the detailed design decisions that are required to implement the target site. For example, the project team will want to document the required interfaces between commands, JavaServer Pages (JSPs), and the persistence layer of Enterprise JavaBeans (EJBs), data beans, and possibly, LOQS objects.

Completion of all design activities enables the project team to initiate the build phase. During this phase, the project team codes the site elements, or components, and when done, the team unit tests each prior to integrating the components into builds. The recommended development strategy a project team should pursue is the continuous and incremental development and code integration testing strategy. This strategy advocates anchoring development on use cases and then testing and integrating code components into builds, which,
in turn, must be tested prior to deployment using a Build Verification Test (BVT) process. The build process must define:

- A build lead who is responsible for managing the process.
- A build schedule outlining when builds will be made available.
- The established roles and responsibilities for team members who create and use builds.
- Build deployment and build failure processes.

In addition to establishing the build process, the project team must also define and implement an integration test strategy. This will enable the team to test and, as needed, to correct code components of builds as they are integrated into the appropriate WebSphere Commerce subsystems.

Extensive details are provided throughout this Redpaper with recommendations and guidance about how to:

- Subdivide implementation responsibilities.
- Structure appropriate BVT and build deployment processes.
- Support error handling after code implementation and site launch.

This Redpaper identifies a systematic strategy for fully testing a commerce site. The site may either be original or one that has been newly migrated from an older site. The objective of the test strategy is to ensure that the customized site components interact correctly with each other, as well as with all relevant out-of-the-box components. The key activities within the site testing phase are:

- Functional Verification Test (FVT)
- System Integration Test (SIT)
- System Verification Test (SVT)
- User Acceptance Test (UAT)

Employing a systematic test strategy streamlines the site testing process, which results in the project team being able to deliver operational site code faster and with improved reliability. To support the site testing strategy effectively, the project team must first create a suitable testing infrastructure, as well as define the testing process. The infrastructure must reasonably mirror the production system as closely as practical, in terms of interfaces with third-party software or system connections, if it is to produce meaningful test results. The testing process will also aid the project team in pursuing the test strategy, allowing the team to complete each test phase, identify bugs, work to correct them, and retest the components. The process should also define how the project team that designed the site may best be divided into the test and problem fixing subteams to support all test activities efficiently.
The actual testing of code may follow either a single-pass test strategy or a multi-pass test strategy. The single-pass strategy advocates that the test team conduct one pass of a test case to determine the code's integrity. This strategy lends itself to projects where there is limited interaction between use cases and test cases. The multi-pass test strategy, on the other hand, requires the project team to conduct two or more passes through a test case to verify the code functions as designed. This test strategy is suitable for projects where there is considerable interrelationship between use cases and test cases.

In terms of the test phases, FVT requires the test team to function test site code from end-to-end using a suite of test cases to ensure it operates as designed. SIT, on the other hand, is an interim test phase, because it enables the project team to assess the test system (using a simulated production environment) to ensure it can and will support SVT and UAT. SVT marks the commencements of the final tests, and it enables the test teams to validate the site code that resulted from the implementation phase. The customer IT team should undertake most of this testing responsibility, because they are best suited to assess the functionality of the site code to ensure it satisfies the stated business requirements. Doing so also prepares the customer team to lead UAT, which requires the testing of site look-and-feel, site flows and behavior, as well as validating business processes. These tests should be performed by the customer's business users.

The key reason for employing a systematic test strategy is to ensure all code operates as designed, and if any bugs are found, that they may be passed to the problem fixing team for prompt resolution. This Redpaper provides numerous details and tips for supporting and pursuing the recommended test phases.

This Redpaper concludes with a summary of the steps and considerations a customer IT team should address prior to deploying the site on a production environment. During this phase of a project, the project team must assist the customer with determining if the production environment is capable of supporting either the new site code or the migrated code. The key activities include:

- Hardware preparation.
- Prerequisite software preparation.
- Hardware and software configuration.
- Implementation of tested and approved code.

In addition to these preparatory activities, the customer IT team is encouraged to perform contingency planning to compensate for potential issues or failures that may result after launch. Core among these activities is the need to define and document how the existing site can be restored, as well as how the new site can be tested, repaired, and re-launched.
Again, the purpose of this Redpaper is to present a viable development method that a WebSphere Commerce development team can use to create or migrate WebSphere Commerce sites. In addition, this Redpaper provides many recommendations regarding team structure; design, implementation, and test strategies; and it reinforces each with real project examples of how to perform specific activities during a new implementation or a migration project.

Additionally, this Redpaper contains a chapter fully dedicated to migrating WebSphere Commerce sites including a discussion of migration strategies, planning and tools.

This Redpaper provides a high-level overview of the available WebSphere Commerce development tools and how each tool may be used. This paper has specific sections that detail unique usage possibilities during key phases of the development method.

To conclude, the key premise of this Redpaper is that tools, when combined properly with a logical development method, can produce significant benefits to a project team and a customer so that either may improve its ability to create or migrate WebSphere Commerce assets efficiently.

1.1 Further information

IBM consultants are available to aid your implementation of the WebSphere Commerce development method as discussed above and in greater detail within the body of this Redpaper.

The IBM service organization is comprised of highly skilled consultants with as broad business and architectural knowledge, as well as deep technical skills, and with access to lead developers and product support personnel in IBM research and development facilities. These service teams provide worldwide assistance and support for WebSphere products to customers and business partners alike through service offerings to aid in e-commerce business analysis, as well as in the design, building, testing, and deployment of applications for e-business.

By engaging the IBM services team, you not only have access to some of the best business consulting, product consulting, and training skills in the world, you also have a team dedicated to help ensure the success of your e-business initiative.

To learn how IBM Business Consulting Services may assist you in the analysis of your e-business strategy, visit the following URL:

http://www.ibm.com/services/bcs/
To learn how IBM Software Services for WebSphere and IBM Global Services may assist you to implement WebSphere Commerce efficiently, visit the following URLs:

http://www.ibm.com/services/igs/
http://www.ibm.com/websphere/developer/services/
Method for project development

In this chapter, we introduce a proven project development method that was specifically adapted for the creation of e-commerce sites using the WebSphere Commerce family of software products. After reading this chapter, you should have a clear understanding of the following topics relevant to the development method:

- Key development phases
- Actions and terms
- Deployment conditions

All subsequent chapters are build upon the overview provided in this chapter.
2.1 Systematic development method

Gone are the days when creating e-commerce sites was considered a fine art requiring highly specialized skills. E-commerce development has become a mature field since first introduced in the mid- to late 1990s. Experience proves that using a systematic method to develop software applications for e-commerce systems greatly reduces project risk and improves proper alignment of software functionality with customer business requirements, two key influencers of project profitability and customer satisfaction.

The key characteristics of a systematic development method are:

- It is systematic.
- The method has been validated and refined through empirical experience gained through e-commerce development.
- The framework of the method is centered around established project management principles to help define work products, deliverables, and time-lines (or phases) to deliver software on-time and within budget, to meet a customer's business needs.

The method outlined in this chapter is flexible to support the creation or migration of both small- and large-scale e-commerce sites. Prior to introducing the method, this chapter first identifies and defines relevant terminology that is used through the remainder of this paper.

This Redpaper covers two scenarios in developing WebSphere Commerce sites:

1. Creating a new e-commerce site; and
2. Migrating an existing version of WebSphere Commerce to WebSphere Commerce V5.6.

Migrating sites from earlier versions of WebSphere Commerce to WebSphere Commerce V5.6 requires careful planning and a systematic approach, for details refer to Chapter 9, "Migrating existing sites" on page 185.

2.2 Definitions

To improve your understanding of the project development method, this section will provide an definition of the central terms used in this Redpaper. This section establishes a vocabulary that:

1. Is used throughout the remainder of this Redpaper.
2. Enables you to use this vocabulary when communicating and sharing knowledge of the method with others.
2.2.1 Work product

Work products are tangible pieces of information or artifacts that can be created, shared, and communicated. The method itself is work product based, meaning that work products are the central piece of information to the method.

The following are all examples of work products:

- An education or training roadmap.
- A macro design document.
- Deployment code.
- An e-commerce database with customer information.

2.2.2 Deliverable

A work product becomes a deliverable when it has been highlighted in a contractual document that has been signed by both the customer and a services provider, such as a Statement of Work (SoW) or a Document of Understanding (DoU).

Deliverables are thus work products that the engagement team are contractually committed to deliver to the customer. Deliverables will typically also have to be accepted by the customer, be it through a sign-off, or an acceptance test.

Due to the overhead of the acceptance process, as well as the potential risk of overloading the customer with information, the number of deliverables should be kept at an acceptable minimum.

**Important:** With respect to the method, deliverables are neither more or less important than non-deliverable work products. During some projects, non-deliverable work products are sometimes overlooked, or even intentionally skipped. This is dangerous, as all work products are equally important during the life cycle of both the project and the extended maintenance period of the final site.

As such, minimizing the number of deliverables should not be regarded as a way of minimizing the number of work products that must be produced.

2.2.3 Phase

The development method consists of a set of time or function bound phases (for example, the design phase). From a system-view, a phase has very clear input and output work products (for example, the customer requirements document).
A phase takes one or more work products and performs a set of activities or tasks to enhance existing work products or create new work products. A phase is generally closed before starting the next one in sequence.

### 2.2.4 Activity

An activity is an element of work to be performed during the course of a project. In the context of a work breakdown structure organized hierarchically by Phase → Activity → Task, an activity is an element of work at a particular hierarchical level.

These activities may be reused in different phases. Examples of activities are:

- Conduct Static Testing (Micro Design phase)
- Perform Programming Cycle (Build Cycle phase)
- Perform Development Testing (Build Cycle phase)

**Note:** Alternatively, the term subphase can be used instead of activity.

### 2.2.5 Task

A task is a subdivision of an activity and is the smallest unit described in this method.

A typical project phase generally involves many tasks, and each may be executed either sequentially or in parallel with one another. In addition, depending on their individual requirements, tasks may be completed by individual team members or distinct teams or subteams.

A task is likely to require input in the form of work products, and once it has been completed, it will produce one or more output work products. For example, when creating a database, you would require a data model for the target database to define and create the database schema. When complete, the target database may then be populated with customer or sample data to create the required database.

As an example, the following is a list of the tasks in the Perform Development Testing activity:

- Review Source Code
- Conduct Unit Tests
- Conduct Integration Tests
- Conduct Usability Tests
- Evaluate Programming Cycle Results
Each of these tasks have defined primary and assistant practitioner roles, as well as work product inputs and outputs.

### 2.2.6 Work unit

Used in early definition stages to help organize and construct the work breakdown structure, work units are grouping of activities that have certain common properties.

A work unit can be used as a reference to any of the following terms:
- Phase
- Activity
- Task

### 2.2.7 Work breakdown structure

A hierarchical decomposition of project activities, including dependencies, into lower-level activities and tasks. Each descending level of this decomposition represents an increasingly finer view of the constituent parts.

### 2.2.8 Strategy

In terms of this Redpaper, a strategy is a method for executing a task. A project team may execute tasks in a variety of ways to produce the desired output work products. Therefore, many strategies may be used to complete specific project tasks and create the associated output work products.

For example, it is possible to use either the migration tools or hand-coded SQL scripts to perform data migration during a migration project. Using the migration tools is one strategy for creating a target database, while using hand-coded SQL scripts represents an alternate strategy. Both strategies should produce the same output work product, namely the target database.

Because a strategy is associated with a task, which may repeat in several phases, its scope may reside within a single phase or span multiple ones.

Each strategy possesses factors that influence its selection by the project team as the strategy for a particular project phase or phases. Factors most likely to influence a project team's decision making process are as follows:
- Resource skill level.
- Business requirements.
- Availability of tools or aids to streamline tasks.
Before pursuing any one given strategy, you are strongly encouraged to assess and compare the advantages and disadvantages of potential candidates to select the appropriate ones given your project's requirements.

2.2.9 Customer

The customer should be viewed as one who either owns an e-commerce site or who wishes to develop an e-commerce site. In either case, the customer defines and approves the business requirements for the e-commerce site.

2.2.10 Customer IT team

A team consisting of one or more members, each assigned by the customer, that is responsible for maintaining the customer e-commerce site and associated computing infrastructure, such as the back-end or legacy systems.

2.2.11 Project team

Owing to its responsibilities, the project team is prominently positioned with the method in this Redpaper, because the method seeks to emphasize the activities of this team.

Because a project team may consist of subteams, such as a design team or test team, the appropriate subteam will be identified by name if it is responsible for a method task. In other words, the project team will be the primary reference team used either explicitly or implicitly throughout this paper unless specified otherwise.

In terms of projects, a project team is sanctioned and supported by the customer during an in-house project, while during a consulting engagement, the project team is controlled and supported by outside services organizations, such as ISSW, IGS, or IBM Business Partners. No matter the allegiance of the project team, all should carefully assess project execution and control requirements whenever team members will be geographically dispersed.

The assessment would typically be based upon evaluation of factors, such as frequency of communication, the quantity and nature of control information supplied, and the difference between expected and observed progress in project execution.

Each project manager or leader will need to understand the working relationships between team members to plan and coordinate project activities appropriately, especially when the teams reside in different locations. This topic is an important part of project management discipline, such as the Project Management Institute (PMI) standard adopted by IBM. However, as further information about
implications of dispersal on projects would be available from a project management specialist, we do not discuss the topic further in this Redpaper.

2.2.12 Project repository

A repository of project-specific information should be established and maintained for all team members to access. It would hold background information, reports, minutes of meetings, results of discussions, and indeed, any other kind of information that is relevant to the execution of the project.

The project repository could be implemented using any workflow management tool, for example, a Lotus Notes database. It may be better to make use of collaborative facilities, such as those provided by Lotus Quickplace, which supports not only team information and resources sharing, but can also support controlled external access by customers using Web browsers.

2.3 Project life cycle

Figure 2-1 graphically represents the six phases of a typical project development method. Within this Redpaper, we focus on the following four core phases:

1. Design
2. Implementation
3. Site testing
4. Deployment

The following figure also illustrates the flow iterations based feedback provided from the different phases.

**Note:** The purpose of this figure is to show the most common phases of a project, this is not an exhaustive list.
The following is a high-level work breakdown structure based on the project life cycle illustrated in Figure 2-1. In this work breakdown structure we only show some common activities, tasks are not shown.

► **Analysis**
  – Business requirements gathering and analysis.
  – Fit-gap analysis.
  – Constraints definitions.

► **Design**
  – Solution outline.
  – Macro design.
  – Micro design.

► **Build**
  – Establish development environment.
  – Establish test environments.
  – Perform programming cycle.
  – Perform test cycle.
- Develop support documentation.

► Site test
  - Functional verification test (FVT).
  - Systems integration test (SIT).
  - System verification test (SVT).
  - User acceptance test (UAT).

► Launch
  - Cutover to production.

► Site maintenance and enhancement

Once the e-commerce site has been launched, any new modifications added to the site after the project ends should be managed as functionality enhancements (shown in Figure 2-1 as *New Release*).

During the lifetime of the e-commerce site, one or more of the method phases may be repeated as necessary to facilitate enhancements (shown in Figure 2-1 as feedback arrows). Typically, repeated implementations of the method will be required to incorporate significant enhancements to the site, such as new versions of WebSphere Commerce software or updates, or to accommodate changing business requirements.

### 2.3.1 Core development phases

Figure 2-1 highlights the core development phases and how those phases tend to be iterative. Following the figure is the work breakdown structure showing the common activities for each phase. It is important to understand that proper assessment of a project will help determine which, if any, of the activities will apply. Also note that relevance and scope of each activity will vary from project to project. To accommodate this, a method adoption workshop is typically arranged during the analysis phase.

**Analysis phase**

The customer must consider, assess, and plan their e-commerce site carefully. Understanding site requirements, defining the nature and intricacies of the existing site for transition projects, and documenting the required features and functions of the target WebSphere Commerce site are key customer activities that should be completed during this phase.

A common event during this phase is for the customer to request estimates of resources required, time to develop, and other details, such as the machine specification required to host the final site. The intention is to use the provided numbers for budgeting or sizing purposes. It may be possible to provide
extremely rough estimates, based on similar previous projects. It would be impractical to provide more accurate figures until later in the project, and the customer must be made aware that any estimates at this phase are tentative.

**Design phase**
The design phase marks the beginning of the WebSphere Commerce project and consists of the following three activities:

- Solution outline
- Macro design
- Micro design

Taken together, these activities represent incremental and iterative design activities that enable the project team to define the target system details. Details of this phase are described in Chapter 3, “Design phase” on page 27.

**Build phase**
During the build phase, the project team focuses on coding the design that resulted from design phase activities. This phase also includes continuous and incremental unit testing and the integration of required software components. Details of this phase are described in Chapter 4, “Build phase” on page 59.

**Site testing phase**
The site testing phase covers several formal activities, all of them enable the project team to test the newly developed or migrated e-commerce site assets systematically. Details of this phase are described in Chapter 5, “Site test phase” on page 89.

**Launch phase**
During the launch phase, the project team assists the customer IT team to bring the e-commerce site to a production-ready state and, as required by the customer, to support and help its IT team to enable the site. Details of this phase are described in Chapter 6, “Launch phase” on page 105.

**Project close phase**
The solution close phase is where the results of the project are assembled and archived for use by other practitioners and analyzed. There is one major activity in this phase that is capture project results. This also the phase in which the project team performance is analyzed and estimates are compared to actual spendings with a focus on improving any shortcomings in later projects.
2.4 Using the method

Using the method effectively involves certain considerations that will be elaborated below. Your understanding of the implications and requirements for each of the several considerations outlined here will improve your ability to employ the method during any given project.

Developing a firm understanding of the considerations will enable you to effectively adapt the method presented here to meet specific requirements of the development work for a given project. For example, the method presented in this Redpaper provides guidelines for deciding between the use of all the design activities (solution outline, macro design, and micro design), or combining one or more activities within a given e-commerce project. Understanding and carefully evaluating these guidelines, and applying the results to an e-commerce project, will increase the chance of the project manager making the best decision for the project, and so increase the probability of success.

2.4.1 Customizing and adopting the method

It is possible to customize this method to allow it to be applied to building all types of e-commerce sites using WebSphere Commerce software. This includes B2B and B2C sites, which may be small, medium, or large in size.

For each phase, this Redpaper describes core strategies that aid the project team to complete key tasks. Approaches for customizing the method are also provided to enable project teams to tailor it to specific project requirements. All the strategies identified in this Redpaper are grounded in best practices that IBM services teams have derived from actual WebSphere Commerce projects and engagements.

2.4.2 Project roles and skills requirements

Any given project will have specific role and skill requirements. In terms of this Redpaper, most of the activities will be described from the perspective of either a project manager or a WebSphere Commerce architect. The purpose of this paper is to enable project managers, WebSphere Commerce architects, designers, and developers to define, plan, and execute WebSphere Commerce projects efficiently by providing appropriate guidance, recommendations, and examples. Common roles and skills needed for typical WebSphere Commerce projects are discussed in detail in Appendix A, “Project team creation and common roles” on page 225.
2.4.3 Structuring information

Because each phase and activity is likely to have a corresponding list of tasks requiring completion, each task may have one or more execution strategy. Some tasks and strategies are discussed in detail, while others only receive cursory mention. Strategy tables are provided to summarize the elements and rationale for each strategy so as to enable the reader to select an appropriate one to complete a task.

Note: The development method may accommodate many strategies, so the reader is encouraged to develop strategy tables, using the templates provided when building project plans. Also, the reader is encouraged to assess and review all strategies first, prior to selecting one to proceed with.

The following recommendations will aid you to develop the appropriate project plans:

► It is very important that the main phases of the project development method should never be ignored or skipped when building a project plan.

► The activities of this method can be combined, overlapped, deleted, replaced with others, or supplemented by new activity to suit project needs.

► The list of tasks and strategies that are specified in this Redpaper represent a subset of the possible tasks and strategies that could be applicable during a project. We would recommend adding new and relevant tasks and strategies when creating a project plan.

► Prior to customizing any activity, fully assess the associated advantages and disadvantages of the customizations. All the tasks and strategies listed in this Redpaper are based upon best practices gained from direct experience.

2.4.4 Case studies

While describing the various phases, activities and tasks of the development method, we provide examples to aid your understanding the phase or task. Many of the examples were inspired by case studies of real projects. The examples provide concrete scenarios for you to use and adapt during your planning efforts.

2.5 Summary

In this chapter, we have defined a project development method that can be used to develop new e-commerce sites or to migrate existing sites built with earlier versions of WebSphere Commerce software.
Having reviewed this chapter, you should be able to:

- Describe the core phases of the development method.
- Identify key development terms and concepts.
- Discuss relevant deployment conditions

In the following chapters, we will discuss and frame the core development phases of the method.
Design phase

The design phase is the first major activity on a project following the analysis phase. The gathering of the business requirements and fit-gap analysis conducted during the analysis phase serve as input work products to the design phase. In addition to technical activities performed by the project team, the project manager, working with technical lead staff and the customer project manager, starts to build a detailed project plan.

This chapter identifies the design activities performed by the project team and the project planning activities performed by the project manager. Each is a core design activity, and in some cases, the resulting output work products will be used throughout the remainder of the project. Also, some of the output work products will need to be completed during this phase prior to the project team initiating the build phase. This phase is discussed further in Chapter 4, “Build phase” on page 59.

Note: The process of developing a commerce site, whether it is a new site or a migration of an existing site, is one that requires the same precision and rigor as any other significant systems engineering project. This chapter frames the development process for you.
3.1 Purpose of the design phase

The purpose of the design phase is to define and document the site design for the target site. The key deliverable of this phase is a detailed project plan (a work-breakdown structure), which the project manager should create before exiting the activities of the design phase, that is, on completion of the solution outline, macro design, and micro design activities.

Note: Irrespective of whether a project involves the implementation of a new site, or the migration of any existing one, if the work is similar to a previous project, or appropriate resources or skills are available, the project manager may be tempted to reduce or eliminate some aspects of the detailed design and proceed with planning later activities.

Doing so introduces a substantial risk that the detailed design activities uncovers or introduces details that will require the earlier plans to be discarded or at least significantly modified.

Such bypassing of the design phase can thus not be recommended!

Once the site design has been defined, and all technical work items are identified, the project manager should add each to the project plan.

While not an explicit activity, we recommend adding lead time to each phase within the project plan. This recommendation is based on good project management practice and practical experience of commerce implementations. Some tasks require substantial planning to ensure that all prerequisites are met. Adding lead time to all tasks is a good practice, because it provides you the opportunity to compensate for unforeseen circumstances when executing on projects.

Note: Defining the data model early in a project is important because it will be used to lay the foundation for the overall site design and subsequent data creation or migration activities. In addition, the process for developing the data model may reveal aspects of the project that were not noticed during prior activities, but that may require adjusts to the project plan.
3.2 The design phase

The design phase is the formal beginning of a WebSphere Commerce development project. Depending on whether the project purpose is to implement a completely new site, or to migrate and existing site, there will be a slight difference in the entry point to this phase, as shown in Figure 3-1.

![Figure 3-1 Collecting information according to the nature of the project.](image)

An overview of the design phase itself is shown in Figure 3-2.

A subset of the project team will be assigned to focus on the design activity. The team must have specialists with skills in WebSphere Commerce architecture and design. These roles are described in more detail in Appendix A, “Project team creation and common roles” on page 225.
3.2.1 Creating the site design

Creating a design for a commerce site is a key activity of the development method. Site design not only influences and controls all development tasks, it also affects the day-to-day use and operation of the site after launch.

Given the importance of the site design, it is essential that the project manager structure design activities in the right way to produce a proper design. After forming the team as previously discussed, the project manager should next determine how to proceed with site design.

Through many engagements, IBM has found two beneficial design strategies, both of which will be described in detail in the following:

1. The iterative and incremental design strategy, where tasks are performed while allowing for the possibility of refining the result by repeating the activity (iteration), and the tasks themselves proceed in a step-by-step fashion (incrementally).

2. The activity combination design strategy. The details for each are discussed in the subsequent sections.
The iterative and incremental design strategy
This strategy divides the main design phase into three activities. The rationale for this strategy is that through close interaction with the customer, the team's understanding of site requirements and constraints will improve over time. The implication is that as the information improves, so will the site's design. This strategy accommodates continually changing information through incremental or iterative design activity.

Note: This approach to the project design and implementation has a high risk of changing the scope and thus the estimates and the final cost of implementation. For this reason, this approach cannot be recommended for fixed prices projects.

Even for time and material projects, it is important that the customer has a clear understanding that this project approach can lead to significant changes in the implementation cost. This is often acceptable, as the customer gets added value, but to avoid misunderstandings, it is important that the customer knows the potential impact that this approach can have on the final costs.

The three activities of this strategy are as follows:

- Solution outline activity.
- Macro design activity.
- Micro design activity.

Each activity builds on the output produced on the previous one.

This strategy promotes interaction with the customer during each activity and encourages formal sign-off as tasks are completed.

An overview of the iterative and incremental design strategy is provided in Figure 3-3.
Experience suggests that the iterative and incremental design strategy is best suited to projects with complex business logic.

**Activity combination design strategy**

As its name implies, the activity combination strategy combines, or alternatively, compresses, the design activities of solution outline, macro design, and micro design into refined design activities. Please note that combining or compression activities does not, in any way, absolve the project team from collecting site information, nor it reduces the amount of information required for proper site design.

The combined approach requires the project team to determine which activities should and can be combined in such a manner as to support the design activities.

The level and details of customer requirements obtained as input to the design phase will aid the project team to determine which combination of activities may be appropriate. For example, given well-defined requirements (that is, requirements that are unambiguous, precise, sufficient, consistent and
non-contradictory), it is possible to combine the solution outline activity with the macro design activity. Conversely, if the requirements are open or not well articulated, work defining the requirements should be performed during the solution outline activity. Only when the requirements have been completed can the project team pursue a combined macro design and micro design activities.

**Note:** The activity combination design strategy is most suitable for simple sites being developed by experienced project teams. If used, the project team should maintain at least two of the design activities, which means the possible activity combinations are either solution outline and macro, or macro and micro. No matter what the final determination ends up being, the project team should engage the customer throughout the design process to ensure the resulting design incorporates all requirements and is satisfactory.

The project team must determine how the design activities can be combined to fully support the design requirements. The customer requirements obtained during the initial project phase will aid the project team in determining how to combine the activities. For example, given well-defined requirements, a project team should combine solution outline activities with macro design ones. Conversely, should the requirements be open, the project team should combine macro design and micro design activities and retain the solution outline in order to refine the design requirements with the customer.

An overview of the combined activity design strategy is provided in Figure 3-4.
Irrespective of the design strategy selected, site requirements should be documented fully prior to the project team commencing with the build phase, as discussed in Chapter 4, “Build phase” on page 59.

### 3.3 Solution outline activity

Prior to starting the solution outline activity, the project manager must ensure both the customer’s business requirements and output from the fit-gap analysis are available and ready for use by the project team. This information will enable the project team to begin its design work. Without it, work on the solution outline is not likely to produce much of value. Should key details be missing, the project manager is encouraged to consult with the customer to obtain the missing details prior to completing the remaining design activities.

In some projects, the need for a solution outline activity may be questioned; it may appear desirable to progress directly to the macro design activity. The reason why the solution outline activity is so important is that most projects will require information to support budgeting, capacity, and skill planning as soon as
possible. More accurate information to help provide these details would not
normally be available until after the completion of the macro design activity,
which may take some time. Addressing these issues as part of the solution
outline activity allows the project team to avoid a delay, which may potentially be
measured in months, and so progress key directional decisions earlier, as well as
allowing parallel activities, such as ordering of hardware.

The major output work product produced during this activity is the solution outline
document. It captures the initial high-level design for the target site.

**Tip:** It is very valuable to have a standard templates for capturing and
communicating the design details.

The tasks and work products used in the solution outline activity are detailed in
Figure 3-5. It is important to note that during each activity, besides the core tasks
listed, additional project activities would typically continue in parallel. The main
tasks are described in detail in the following.

![Figure 3-5 Solution outline activity.](image)

During the solution outline activity, the project team will lead substantial
interactive discussions with customer to help define the design requirements. It
will normally take several iterations of technical investigation and discussion to fill
in details of the requirements, define external system and component
dependencies, and finally, deliver the initial high-level design that incorporates
each requirement. Iterations are often necessary to clarify points of
misunderstanding, to supplement detail where more information is required, and
to address omissions or agree to late additions to the requirements.
3.3.1 Gathering the functional requirements

The first step in the solution outline activity is to gather the functional requirements, which are driven by the customer's business requirements for the site.

**Note:** Although projects have both functional and non-functional requirements, we will focus on the functional requirements in this Redpaper. This should not be seen as an indication that non-functional requirements are unimportant. Non-functional aspects, such as security, scalability, maintainability, etc., are very important parts of any project, WebSphere Commerce-based or not.

Functional requirements for an WebSphere Commerce project normally consist of the following:

- **Use cases.**
  
  Use cases is the basic way to capture functional requirements. Due to the nature of WebSphere Commerce projects, most use cases will be similar across projects, describing standard WebSphere Commerce functionality, such as the shopping flow. This by no means imply that such use cases can be left out, since use cases describe the functionality of the site and this forms the foundation for the test cases used in the testing phases.
  
  In effect, the use cases describe how the system should work. Without the use cases, it is not possible to formally validate if the system works as required.

- **Site flow descriptions.**
  
  The site flow descriptions document how the user is directed through the site, and can thus be viewed as a high-level use case model.

- **Business rules.**

- **Integration with other systems or products.**
  
  This will typically be captured in the System Context Diagram with its accompanying description.

Some customers will already have one or more of these defined. Others will depend on the project team to help them come up with the functional requirements.
For most WebSphere Commerce migration projects, the project team is typically given the customer's current site as the starting point for building the functional requirements. The project team should start by reading the customer's existing requirements documentation, reviewing the current site, and reverse engineering the code as appropriate. Refer to Chapter 9, “Migrating existing sites” on page 185 for specific WebSphere Commerce migration projects considerations.

If several WebSphere Commerce designers are working on the solution outline activity, the project manager may wish to divide the design team to enable its parts to focus on design activities relevant to specific WebSphere Commerce subsystems. No matter on what part of the design a designer is working, a subsystem or whole, they will be responsible for the following activities:

- Creating a comprehensive and complete set of use cases.
- Creating site flow information and diagrams.
- Determining the required business logic and processes used for site operations.

It is extremely helpful to have an internal project bulletin board, typically as a part of the project repository, where each team member can post questions and responses about the project. During the solution outline activity, these questions will help resolve ambiguity about functional aspects of the requirements, unclear business rules, and inaccessible paths through the site flow.
Tip: This sort of a bulletin board can be useful even for a small team, in which case the bulletin board is used mostly for capturing design issues and decisions before they are finalized in the design documents. This type of historic decision documentation can be helpful in later maintenance or functionality enhancement projects.

Note that such inter-team communication features cannot replace team brainstorming meetings. Team meetings where open discussion is encouraged can often help mitigate design issues, although too frequent meetings can become counter-productive.

On a regular basis, the project manager should review outstanding team questions and discuss them with the customer. Depending on the number of questions, weekly reviews may be sufficient. Sending the questions to the customer using e-mail means that they are written down in a form that allows the customer to discuss them internally, and as necessary, for the customer’s project manager to produce an answer for the project team. Communicating using e-mail as a medium can also help to emphasize ambiguities as people often more easily recognize such ambiguities in the written medium than in oral form.

Note: Delays in answering core design questions are surprisingly common in projects. The questions normally arise from ambiguities or inconsistencies with the customer’s business requirements, which may indicate similar uncertainty or difference of vision in the customer. Resolving these problems and answering the questions may take time until an agreement is reached, and it is possible that the project development team may be invited back to the customer to provide additional help. Delays of this type must be incorporated into the project plan.

Periodic communication with the customer will often result in the project team developing direct contacts with core members of the customer IT team. This is a potential windfall, as it allows an efficient channel for sorting out technical problems and ambiguities.
Once the fundamental requirements have been gathered, the project manager should arrange a workshop with the customer for the project teams to confirm the requirements.

The solution outline activity deliverables include:

- The functional requirements.
- The non-functional requirements, who we will not discuss in detail in the Redpaper.
- The solution outline document.

The functional requirements and the solution outline document are an extension of the customer's business requirements, describing how these will be realized. The functional requirements transform the business requirements into more tangible site specifications and implementation details. The solution outline document, on the other hand, enables the project team to define how each business requirement can be mapped to either out-of-the-box WebSphere Commerce functions or specific customizations. The document should also address how the proposed site will integrate with existing legacy systems, third-party applications, or back-end systems.

### 3.3.2 The solution outline workshop

During a solution outline workshop, the design team will validate all functional requirements and review targeted out-of-the-box WebSphere Commerce functions with the customer team. The objective of the workshop is to ensure the proposed site uses all available features provided by WebSphere Commerce. The customer and design teams must have a common understanding and shared vision for the planned site.

An overview of the solution outline workshop is shown in Figure 3-6.
The design team starts the workshop by inviting the customer team to talk about their current site structure and any ideas for future expansion plans. The design team then uses the information to identify how WebSphere Commerce can be used to fulfill the customer's stated needs. The design team will make every attempt to stress as much out-of-the-box WebSphere Commerce function as practical over product customizations, but where necessary, it will explain how the product should be customized to suit unique requirements. The workshop discussion is likely to reveal potential third-party application or legacy system dependencies.

Product functionality is assessed through WebSphere Commerce subsystems. The design team should conduct a product demonstration stressing the functionality provided by each relevant subsystem. This review will also enable
the customer to fully understand the capabilities offered by WebSphere Commerce.

Following the review, the design team should walk through the proposed site flow, stressing customer requirements and related business rules. For migration projects, it is very helpful to discuss the current data model that may be extracted from SQL statements and the approaches that may be used for data migration.

Another important topic that is often overlooked is the content management process, which includes both catalog data and static content data. This discussion should cover both initial data load and subsequent regular planned updates. For further details refer to WebSphere Commerce V5.4 Catalog Design and Content Management, SG24-6855-00.

At the conclusion of the workshop, the design team, with review and approval by the customer, should have the business and functional requirements fairly well defined, along with the project scope and any key dependencies. These decisions are then documented as the deliverable from the solution outline activity together with a preliminary design to meet the requirements.

### 3.3.3 The solution outline document

The solution outline document will typically have three sections:

- **System overview and architecture.**
  
  The system overview and architecture captures the site component overview and component responsibilities that together provide a template of the proposed site construction.

- **Architectural decisions, grouped by subsystem.**
  
  The architecture decisions describe the business requirements and assumptions, document the issues encountered and their resolution, and record the high-level design decisions made for each requirement, such as which out-of-the-box function will be used and what customizations will be required. All issues must be discussed with the customer during a subsequent solution outline review period.

- **Outline of assets to be built, grouped by subsystem.**
  
  The outline will normally be used to size projects. For example, it can be used to estimate the required number of JSPs, custom commands, or EJBs. In addition, development of other utilities, such as data loading and integration, should be included in the outline. This information may be captured in a spreadsheet, along with estimates for difficulty rating and sizing.
3.3.4 Analyzing current site assets

Where an existing site is being migrated, it is essential to get a detailed understanding of the current site assets. This involves analyzing the current site assets, such as EJBs, JSPs, and custom database tables or extensions. This analysis must encompass the explicit customer business and technical requirements as part of creating a high-level design for the new site.

We recommend using the migration tools that are included with WebSphere Commerce V5.6. These tools will help migrating instance configuration and standard database tables.

Information included within a fit-gap analysis document can help identify the areas of site functionality that may be implemented using new out-of-the-box functions; the benefit of relying on out-of-the-box functionality is that it helps reduce the quantity and complexity of custom code in the site, thereby reducing both the amount of initial development, as well as minimizing the short and long term maintenance required. It may be necessary to reuse the existing site design if no out-of-the-box function meets the customer business needs for one or more components of the site.

Note: Even if a decision is taken to reuse existing site design for one or more components of the site, we would still recommend using the WebSphere Commerce framework in the new implementation to make sure that the custom design integrates well with the product architecture and benefits from best-practice guidelines.

3.3.5 Design-based sizing

Sizing for the project is usually possible at, or soon after, the completion of the solution outline activity. An earlier, less accurate sizing will already have been performed as part of the analysis phase.

In brief, for each functional requirement, an asset count is produced. This will include the expected number of JSPs, the number of new or extended commands, and the number of EJBs and data beans. Each of these assets is given a complexity level. Next, the effort to develop an asset is established, based on information from prior completed projects, and adjusted to reflect team experience and other pertinent factors. The result is an estimate of the effort required for building and testing each of the assets that can be aggregated with the other estimates to produce a reasonably accurate project sizing.
3.4 Macro design activity

The main focus of the tasks during this activity centers on creating an WebSphere Commerce design, the details of which are recorded in a macro design document.

**Note:** The design team that completed the solution outline activity should also follow through with the macro design for the target site. You may need to consider adding or removing members of your design team to balance resource availability against the time required to deliver the site.

An overview of the work products and tasks for this design activity is provided in Figure 3-7.

![Diagram of Macro design activity](image)

**Figure 3-7  Macro design activity.**

3.4.1 Methods of approaching design activities

There are several perspectives that may be taken when considering WebSphere Commerce site design:

- The overall system level, which involves WebSphere Commerce interacting with external back-end or legacy systems.
The subsystem level, which includes the main WebSphere Commerce subsystems, such as Member, Order, and Catalog. This also includes the supporting infrastructure, such as the database and messaging systems.

The use case level, which includes functional sequences that together form the overall site flow and behavior. Use cases will normally address visual aspects, such as the flows from page to page within the site, and non-visual resources, such as the expected operation and behavior of the order subsystem as it communicates with the back-end inventory system.

The component level, which includes JSPs, data beans, access beans, EJBs, and commands.

The design team should consider each of these levels during the design phase for the main site. Considering use cases as a means of site design is especially appropriate during the macro design activity. In a use case strategy for site design, the design team creates detailed use cases. This approach is also consistent with applying an object-oriented method to site design, because it helps to identify the actors and actions performed within the system.

Note: Driving an overall design strategy through use cases yields significant advantages, especially during the build and site test phases, which are discussed in Chapter 4, “Build phase” on page 59, and Chapter 5, “Site test phase” on page 89 respectively.

An overview of the use-case-based design strategy is provided in Figure 3-8.
By centering the design around use cases, we mean that all the design decisions made during the macro design activity will categorize site components, such as commands, EJBs, and JSPs, based upon use cases for the target site.

**Note:** Since the use cases are so central in the use-case-based design strategy, this also makes it very important that all functional requirements are documented through use cases. An often overlooked category of use cases, are those that describe how the WebSphere Commerce system communicates with external systems.

One way of applying the use case design strategy is to create a macro design document template that the design team can use to create the WebSphere Commerce site design. This template should structure information such that appropriate levels of design decisions, such as “What JSPs, commands, and EJBs are needed to implement the user registration use case?”, are based on use cases for the target site.

A major output work product from the macro design activity is the interaction diagram for each use case, described at command and JSP levels. These
diagrams help clarify the site objects and components. For example, clear identifiers are assigned to the objects, and the diagrams show how each object is called or accessed when performing a given WebSphere Commerce task.

**Note:** Large projects may contain several hundred use cases. In such projects, interaction diagrams are not needed for each use case, but can be produced for the architecturally significant use cases. Care should always be taken when deciding not to produce interaction diagrams for all use cases, however.

### 3.4.2 The data model

Another major output work product from the macro design activity is the definition of the data model for the site. Creation of a valid data model for a new commerce site requires a thorough understanding of the customer data and its structure, as well as detailed knowledge of the out-of-the-box WebSphere Commerce data model.

**Note:** The site data model is a part of the development work that, in some cases, does not receive as much attention as other more visual aspects, such as the Web pages or site flow. The low visibility associated with development of the site data model may result in an underestimation of effort required for data model development. The effort to create the data model may be increased both by the complexity of the site data and also by the large quantity of data to be processed. It is, therefore, very important to start working with real customer data as soon as possible. Even subsets of customer data may be used to test the validity of the development database.

The first step in defining the data model is for the design team to identify the data entities from the business requirements and then understand how the data is used and how it flows through the site. For WebSphere Commerce sites, it is useful to categorize the data by WebSphere Commerce subsystems:

- **Member data.**
  
  Data that is related to user configuration, organizations and security-related information, such as access control data.

- **Marketing data.**
  
  Data for campaigns, advertisement, coupons, discounts and collaboration.

- **Inventory data.**
  
  Information about inventory levels and the way that the site should manage the inventory.
- Trading data.
  Pricing information, except discounts which is handled by the marketing subsystem, auction and contract information

- Catalog data.
  Product data and categorization.

- Merchandising data.
  Cross-sell, up-sell and other catalog entry relations.

- Messaging.
  Data for the messaging subsystem, for example transport configuration, definition of message types, etc.

- Order data.
  Order related data, including shipping costs and taxation.

For example, user data is first created when the user accesses the site as a guest shopper. They eventually become a registered shopper with a login identifier and password. Typically, a registered shopper is assigned an internal customer number by a back-end system.

By following the data flow in the above manner, and then analyzing the data used at each stage of the flow, the design team should be capable of ensuring that the data model contains all required data entities.

For each of the data types, the data can be split into configuration and operational data. For example, the order system contains both information about the actual orders, which change rapidly during the lifetime of the site, as well as configuration data about shipping and taxation rules, which change less often.

The focus of catalog data is on the purpose of the catalog, how the catalog is structured into a sensible hierarchy, what the product information consists of, and how content, such as images and support documents, are related to the main catalog data.

The following step is to use the functional requirements to come up with the data structure necessary to support the business logic. This data is distinct from the catalog data, because it is not part of the data to be presented to shopper; instead, it is related to the business rules required for supporting the functional requirements.

An example of data-supporting business rules would be to use product sets with contracts in order to apply constraints on the catalog navigation, such that a customer would have a restricted view of part of the catalog and products, depending on the product brand information they are authorized to see. Similarly,
shipping rules may be affected, where certain products cannot be shipped across international boundaries, or have a limited shelf life, excluding a slower shipping method.

In contrast to new site projects, the data gathering task is much simpler in migration projects, because it usually requires analysis of the existing data model and understanding the usage of each data field.

After gathering all the data required, the design team should compare the required site application data model with the out-of-the-box data model. In most cases, the out-of-the-box data model will be sufficient.

**Note:** It is especially important to ensure that data stored within the standard out-of-the-box tables will be used exactly as specified in the data model. Misunderstanding the standard data model and the consequent effect of standard commands on data stored in the model can have deep-seated and hard-to-reconcile consequences for the project.

For example, in some multinational organizations, a single product may be offered under several different brand names. It may be tempting to treat the brand as simply an attribute of the product and store this brand information within the standard out-of-the-box ATTRIBUTE and ATTRVALUE tables as descriptive product attributes. However, this might introduce other problems or complexities, because the brand may also be used to support catalog navigation logic and, therefore, the range of products a customer can view. In this case, the brand is not merely a product attribute, but also has a significant effect on the perceived structure of the catalog. In such cases of incompatibility between the stated purpose of an out-of-the-box data model component and the desired application, it may be better to design new tables that more effectively support the desired logic.

Not all site-specific data requires a new custom table. Almost all WebSphere Commerce tables provide some custom fields, but very little logic related to them except direct support for read and write. If the extra data falls into this simple category, it is sensible to use the custom fields from out-of-the-box tables. Refer to 8.3.1, “Customizing WebSphere Commerce tables” on page 147 for more information about utilizing the standard WebSphere Commerce database schema.

Something that should definitely be avoided is adding new columns to out-of-the-box tables to support extra data. These extensions would mean that the standard EJB for the table would not cover the new fields. It would be much better to use a custom table to store the extra data, with a primary index corresponding to the primary index as the out-of-the-box table. A new EJB would be required, corresponding to the new table, but this would be a project-specific
extension and, therefore, would not compromise future migration. For more information, refer to 8.3.2, “Customizing WebSphere Commerce EJBs” on page 148.

**Note:** After the data model has been defined, you are advised to reassess the solution outline and functional requirements to make sure that all the selected out-of-the-box functions identified are correct for the functional requirements, that the out-of-the-box data model supports the site data, and finally, that all the functional requirements that are not supported out-of-the-box have been included in extension data tables.

### 3.4.3 Creating interaction diagrams and class diagrams

Interaction diagram and class diagram are both use case driven. For each use case, there should be at least three possible paths of execution:

- The normal or standard path.
- An alternate path.
- An error path.

Each interaction diagram should address all three paths for each use case. Creating these diagrams will help illustrate the site flow and so assist with the underlying implementation details.

For WebSphere Commerce projects, the majority of classes used will be commands, EJBs, JSPs, and data beans. In terms of macro design, emphasis will normally be on commands and JSPs. Consideration of EJBs and data beans, which address how data is accessed within commands and JSPs, is normally left to micro design.

Having established the site flow for the use cases, the design team should identify the Web pages used for each path. These pages may be static HTML or dynamically generated from JSPs.

The next step is to determine which command to use to complete the flow. By analyzing the use case and referring to the solution outline, it is possible to determine which out-of-the-box command could be used. This involves deciding the following:

- Whether the out-of-the-box command will perform exactly the desired task.
- What each path’s input, condition, and result is.
- What tables are to be accessed.
Based on the analysis, the result is a list of commands that includes standard out-of-the-box commands, as well as the ones that need to be developed as part of the project.

Migration projects have another factor that should be taken into consideration. If any of the original site implementation used out-of-the-box commands, it is likely that corresponding commands can be used for the migrated code. Further, even if the original site required a customized command to be created, it does not follow that the new site will also require a new customized command; the extra functionality offered within the newer version of WebSphere Commerce may provide the desired functionality. It is advisable to repeat the analysis to determine if there is a suitable out-of-the-box command, or what alternative customizations may be needed. Refer to Chapter 9, “Migrating existing sites” on page 185 for further details on migration planning.

Real projects have identified five scenarios for using out-of-the-box commands:

- The standard out-of-the-box command fully meets the requirements.
- The out-of-the-box command must be extended with extra logic, for example, to read or write data stored in a new customer table.
- The out-of-the-box command implementation must be replaced.

**Note:** Replacing out-of-the-box commands must be handled carefully. Other standard out-of-the-box commands may refer to the replaced command. This must be taken into consideration when determining the functionality of the new command logic. This potential ambiguity will not arise if the replacement command has a different name to highlight its project-specific nature.

- The functionality requires that one or more out-of-the-box commands are invoked in sequence.
- A completely new command must be created.

This information should be documented in class diagram. The WebSphere Commerce Information Center contains examples of these development tasks.

JSP can be categorized into five groups:

- JSPs used for display of data.
- JSPs used to allow a user to enter data.
- A combination of data display and data entry.
- Error pages.
Hidden JSPs used to construct outbound messages, either XML, HTML or text, to be sent through one of the outbound adapters, such as the JMS, e-mail or Web Services adapters (this JSP does not play a part in site flow).

JSPs used for displaying data might include HTML files to display static data, in which case, the HTML page should be documented as well. An error page can contain just the error message, or it may be the same JSP that handles the non-error case, including code to display the error message prior to other content. Again, all error conditions and messages must be documented.

After all commands, EJBs and JSPs have been identified, the design team can build the interaction diagrams. The diagrams documents the interactions between the user, commands, EJBs and JSPs. The interaction diagram should define input and output parameters.

**Note:** One of the starter store models is a good basis to start building the customer site. The store models use a well-established design that covers the basic site functions and demonstrates how the commands and JSPs are linked together to accomplish a task. The WebSphere Commerce Information Center contains business process definitions for the starter stores. Refer to 8.1, “Basing development on a starter store” on page 144 for more information about the starter store models.

### 3.4.4 JSPs templates

The design team should create a template for all site JSPs to use to promote a consistent look and feel. This presentation style will usually be designed by the customer. The project team has the task of transforming the “look” into templates for later JSP development. Refer to 8.4.5, “Use an appropriate includes mechanism” on page 165 for tips about establishing a consistent look and feel through the use of JSP include techniques.

**Note:** Customer expectations for site look-and-feel generally exceed basic branding, and the design team must work to manage these expectations.

### 3.4.5 Creating a WebSphere Commerce site database

A WebSphere Commerce site clearly needs a database to store persistent data. The development project for the site must also have a working implementation of a site database for this purpose. In a migration project, the database structure will already be known. The migration tools included with WebSphere Commerce V5.6 should be used to migrate such a database so that it will be complete at the end of the macro design activity. For new sites, it would be expected that at least
a high-level view of the database would be available after the macro design activity, sufficient to implement many of the required data structures, but still likely to be refined following the micro design activity. For details on the available development and migration tools refer to Chapter 7, “Development and migration tools” on page 115.

Table 3-1 shows the expected status of database design work within each design activity, according to the type of project being executed.

<table>
<thead>
<tr>
<th>Table 3-1  Database status for design activities, according to project type.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project type</strong></td>
</tr>
<tr>
<td>Idealized project</td>
</tr>
<tr>
<td>New implementation project</td>
</tr>
<tr>
<td>Migration project</td>
</tr>
</tbody>
</table>

Creating or migrating a sample database by the end of the macro design activity has several advantages.

During migration projects, the database will serve the design team when they translate and test target site SQL statements to determine if session beans are appropriate. A prerequisite for creating a sample target database is the finalized data model that is produced during the macro design work. The persistent data layer, which consists of out-of-the-box and customer EJBs, access beans, and data beans for the target site, will also require the final data model. Finalizing the data model for the target site enables the design team to create a sample, or migrated target database, which in turn, will aid their completing the remaining design details.

Typically, it is the data creation or migration expert who creates or migrates the sample database, for details on user roles refer to Chapter 9, “Migrating existing sites” on page 185 and Appendix A, “Project team creation and common roles” on page 225.

Creation of databases is an activity that applies during both the design phase and the build phase, and the activity holds true for both new and migration projects.
projects. We have already indicated that, ideally, a development sample database would be created during the macro design activity. This database can then be extended to become the solution database during the build phase. This approach minimizes the effort needed to create the solution database.

Each database has a common starting point, which is the out-of-the-box database provided by WebSphere Commerce V5.6. This database provides the foundation for the out-of-the-box functions. The foundation consists of the schema, table constraints, bootstrap information, and indexes. If no further customizations are required, it should be possible to load customer data directly into the database early in the project. If further customizations are required for the site, the new schema, constraints, and indexes must be applied to the original database prior to loading the data.

**Note:** The sooner the design team obtains customer data, the better for the project. It is surprisingly common in projects that the customer supplies data that is supposed to be correct, consistent, and complete; yet on closer inspection, it is found to have errors and inconsistencies that may require significant cleansing. Such problems are typically encountered where the data is owned by or originates from different departments or divisions.

Figure 3-9 provides an overview for two different strategies for creating the database, depending on whether the database is for a new site project or a migration project. The strategies are different, although the initial steps required to create the solution database are the same. An important factor that affects the success of both strategies is the separation of work into their subsystems and assigning the more difficult ones to the more experienced developer. Separation of duties supports code maintenance.
Details on how to migrate an existing database are covered in Chapter 9, “Migrating existing sites” on page 185. The available tools are covered in Chapter 7, “Development and migration tools” on page 115.

### 3.5 Micro design activity

The micro design activity follows the macro design activity. The purpose of this activity is determining what part of the data access layer should be implemented with entity EJBs and what part with session EJBs, as well as to determine the detailed interface between commands, JSPs and EJBs. For this activity to proceed smoothly, the design team must have a sample target WebSphere Commerce V5.6 database, whether it is a new database, or migrated from a previous WebSphere Commerce version.

One of the main goals of the third iteration of the design phase is to build on the macro design by capturing the detailed design decisions. The capture of detailed design decisions is a key requirement needed to start the build phase. The design decisions should be recorded in the micro design document, which the
customer should review and approve, that is, obtain appropriate sign-off on the design details.

A core micro design task is to define and create the appropriate interfaces between commands, JSPs, and the persistence layer of EJBs and data beans. Pseudo-code, representing business logic in commands, should also be recorded in the micro design document.

An overview of the tasks and work products of the micro design activity is provided in Figure 3-10.

Before proceeding to the next phase, the design team should verify that either the database has been migrated, or that a target database has been created for the project, as indicated in Table 3-1 on page 52. As determined by existing technical conditions, the database design might require updates should the data model be changed after it has been reviewed with the customer during the micro design activity. No matter what happens during micro design, all database creation tasks must be finished before the end of the micro design activity. Failure to produce a valid database at the end of the micro design activity will impede progress, because workarounds and delays will be required to achieve code requirements during build phase.
Source SQL statements from earlier sites collected in the previous phases should now be analyzed to determine which ones may be replaced by suitable data beans. Entity EJBs are best suited to replace statements with data read or data write requirements, and session EJBS should be used to generate read-only beans for statements requiring read-only access. SQL statements that are candidates for entity EJBs need to be mapped to existing EJBs or implemented as new custom EJBs.

Before proceeding to the build phase, verify that the database has been migrated and is available for use by the coding team. In addition, the project team should create a team development environment that uses a shared Source Control Management (SCM) team repository server and the common (migrated) WebSphere Commerce V5.6 database.

**Note:** Although it is recommended to use a shared database, some team members may be involved in activities that either requires a very stable database, which would be the case for performance testing activities, or a highly volatile database, which would be the case for as team members working with frequent deletion and reload of data, for example migration script testing.

Such team members should have access to dedicated databases, which could still be located on a centralized database server, if required.

The micro design activity should produce a list of line items to implement as part of the detailed project plan. If a site migration project is being performed as part of a consulting engagement, it is advisable to complete a second statement of work to document all project execution activities that will be required through to completion. The test strategy that began during the early stages of the project should evolve into a more detailed test plan.

### 3.6 Design review

The design review is the first major opportunity afforded the design team to review the match between the main site objectives and the site's design to meet the objectives.

A good design will have certain characteristics:

- It should be focused on producing a site that will apply and demonstrate best practices.
- It should maximize the incorporation of reusable components, or where new components are designed, there should be consideration for future reuse.
The design should comply with the spirit, as well as the letter, of the WebSphere Commerce programming model.

When reviewing a design, the design team should consider the design's context, especially so when an existing site is being migrated. Earlier versions of WebSphere Commerce used different implementation architectures and technologies. In addition to migrating the site, the customer must also invest in new tools, education, and techniques that will be needed to support the migrated site.

Note: Generally, customers strive to reduce the number of platforms and technologies requiring direct support. Therefore, many customers will view favorably the opportunity to reduce costs when optimizing their existing site.

During migration projects, project teams focus on specific success factors, such as:

- No change to database schema, and only limited changes to database access plans, as optimized by the customer database administrators.
- Minimum impact on existing back-end and supporting infrastructure.
- Minimum increase in maintenance and retraining costs.
- Maximum preservation of the shopper experience; unnecessary changes should be avoided.
- Maximum, on-going parallel work to avoid a moving target situation.
- Careful analysis of previous, current, and predicted database use and needs to come up with an appropriate data services architecture.
- Optimum scheduling, use of available team skills, and project management.
- Appropriate value-add to the WebSphere Commerce technology.

### 3.6.1 Data analysis review

The data model proposed for the site should be reviewed carefully. Even large projects should require only few additional tables.

Some tables will require transactional reads and updates. In those cases, Container Managed Persistence (CMP) EJBs work best. They require less development work and, by default, will provide optimized performance.

The majority of tables are likely to require read-only access. Many of them contain data that changes infrequently, if at all. In such cases, use of CMP EJBs is inappropriate, because the site will not benefit from the advantages afforded by CMP.
The main reason for not using CMP EJBs is that they will:

- Increase the number of database accesses, proportional to the number of tables accessed and rows retrieved.
- Increase the network traffic; more SQL is used, and the results will return whole objects instead of 1 or 2 columns.
- Increase resource management demands, to support tasks such as passivation and memory reclamation.

**Note:** In a very large project that required over 450 SQL statements:

- Nearly 300 of those statements accessed a single table.
- Just over 80 of those statements involved a join of four or more tables, many of which were out-of-the-box WebSphere Commerce tables.

Through experience, IBM Software Services has learned that there is no correlation between the number of SQL statements required and the project's data model. The number of statements reflects the way the logic of the application is built, optimized, and tested.

To minimize the number CMP EJBs used for such tables, use the techniques mentioned in 8.3.7, “Using session beans” on page 152, potentially combined with use of the LOQS tool (see 7.3.1, “Lightweight Object Query System” on page 141).

### 3.7 Summary

In this chapter, we discussed core design activities for project teams creating new sites or migrating existing ones. For each activity, we summarized how it should be performed, what roles and team structure would best suit it, and provided a high-level overview of the skill requirements.

We then described the major phases of the design work, including detailed information about the design of the site data model. In the next chapter, we explain how the design details created during this phase can and should be used to implement the site.
Build phase

In this chapter, we describe build activities of the development method. The core activity during the build phase is application coding, which may be performed using different strategies. We discuss work assignments for the project and build teams. In addition, we introduce initial testing procedures for assessing site code. We also explain the appropriate steps for creating an integrated test environment.
4.1 The project build phase

Following the successful completion of the micro design activity, the project team may now begin developing site code using the specifications produced during the design phase. It is assumed that the project manager will staff the project team with suitable developers and skills, as defined in the project plan and Appendix A, “Project team creation and common roles” on page 225. Figure 4-1 provides an overview of the build phase.

![Figure 4-1  Build phase overview.](image)

Figure 4-2 summarizes the activities and work products produced during the build phase.
4.2 Build and team work assignment

We begin with a discussion of a strategy for build and team work assignment, based on continuous and incremental design. The strategy is based on best practices that have proven successful on various commerce projects. The build and team work assignment strategy specifies that newly implemented project elements, such as JSPs, commands, or EJBs, be unit tested and integrated incrementally. This strategy differs noticeably from the traditional waterfall model, where each major activity follows a sequential development approach. In other words, the site components are coded, then unit tested, and finally, integrated.

4.2.1 Assigning team members to tasks

Prior to starting the build phase, the project manager should complete the following two tasks:

- Select a team organization structure, as discussed in Appendix A, “Project team creation and common roles” on page 225.
- Define the model for producing site code that is test ready, preferably site level.
Figure 4-3 provides an overview of a continuous and incremental development and code integration testing strategy.

![Continuous and Incremental Development and Code Integration Testing Strategy](image)

Even though the continuous and incremental development strategy is very useful, it must have a solid anchor point. In 3.4.1, “Methods of approaching design activities” on page 43, we advocate creating a use-case-centered design. Through experience, we have found that anchoring code development on use cases produces tangible benefits.

Use cases and code components, such as JSPs and commands, can be organized so that the project team start work item assignments and then monitor dependencies within and between use cases. An example of the team organization and work assignments is shown in Table 4-1.

<table>
<thead>
<tr>
<th>Code category</th>
<th>Component name</th>
<th>Use case 1</th>
<th>Use case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSPs</td>
<td>Login.jsp</td>
<td>John</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Place-order.jsp</td>
<td>Joe</td>
<td></td>
</tr>
</tbody>
</table>
A key factor in executing this strategy is the team structure, as discussed in Appendix A, “Project team creation and common roles” on page 225.

If the team structure is based upon the core-competency strategy, as described in Appendix A, “Project team creation and common roles” on page 225, the project manager should assign use cases and most or all of their components to skilled WebSphere Commerce developers to produce the required code. This strategy enables the developer to code, unit test, and integrate code elements for each use case prior to moving to the next use case. The effect of this assignment model is that the use case columns illustrated in Table 4-1 on page 62 will have only single names; only one person would be implementing the complete use case.

On the other hand, if the team structure is organized as using the holistic strategy, Table 4-1 will have multiple names in each column for each use case.

Alternatively, the project manager may decide to use a combination of core-competency and holistic strategies. In that case, the columns of an assignment table would contain a mixture of names. Some columns would have a single name, while others would contain multiple names.

When assigning developers to use cases, the project manager must ensure critical use cases are identified first and that all dependencies are highlighted for the team. This activity enables the project manager to prioritize and sequence the development of code components, and it affords them the opportunity to monitor progress.

### 4.2.2 Tracking progress

Progress may be tracked in many ways. For example, one method assigns completion percentages to specific build stages:

- A 50% measure might indicate the percentage of total effort that is required to complete component coding.
- A 30% measure might indicate the percentage of total effort required to integrate the use case components.
- A 20% measure might indicate the percentage of total effort required to complete unit testing.
Even such a simple strategy should aid the project manager track team progress throughout the build phase.

Important: Extreme care should be taken when tracking progress for a given task. Developers are likely to only consider core development effort when communicating the progress of a given task, overlooking the unit testing and inter-component integration effort. This is not laziness on the developers’ part, but a natural result of each developer’s personal focus on performing the coding effort.

### 4.2.3 Regular builds and systematic integration testing

Once a use case has been implemented, that is, coded, integrated, and unit tested, its components should be published to a test server and retested. Integration testing each use case within the WebSphere Commerce test environment should be considered an initial step in code testing. The task of publishing implemented use cases to a test server for additional testing improves the project team's ability to integrate use cases efficiently. Publishing and retesting use case code is discussed in more detail in 4.5, “Subsystem integration test activity” on page 82.

While progressing through the build phase, the development team should publish regular code base versions and builds to the test server, as described in 4.3, “Integrated development and test infrastructure” on page 65. This activity supports incremental code development, integration, and testing. The team should also create a Build Verification Test (BVT) process, one where each new build is executed before making this build available to the team. A BVT covers one or more test cases or scenarios, and it should test the critical paths through the code that are operational and require validation. Executing the BVT will ensure that the new build has no obvious errors. The BVT process should start as soon as one or more use cases are ready and has been tested on the test server. In addition to orchestrating the build process, the project manager should arrange periodic code reviews with the team to ensure code quality is acceptable and as desired.

The back-end test system should reasonably mimic the interfaces and communication systems of the production systems to ensure the code can be
tested using end-to-end scenarios. The better able the team is to create an accurate test environment, the more effective and thorough its tests will be.

As seen, the first serious testing of site code begins during the build phase.

### 4.3 Integrated development and test infrastructure

To be effective, a project team needs both a development and a test environment. Although each is distinct, they are inter-related and connected.

The environments are as follows:

- **The development environment.**
  
  This environment is used by the project team to develop and share software between team members and then to unit test this software before it is integrated to create a coherent e-commerce site.

- **The test environment.**
  
  This environment is used by the project team to integrate all code pieces and then to test the integrated units, as described in 4.5, “Subsystem integration test activity” on page 82, and 5.2, “Functional Verification Test (FVT)” on page 96.

The physical hardware to implement the development and test environments could be shared. Prior to the start of the build phase, all the required hardware and software, and the team development environment and site test infrastructure, must be ready and available. The project team will require both the development and test environments during the build phase to develop and test site code.

Figure 4-4 provides an overview of a strategy for creating an integrated development and test environment. This strategy is based upon best practices, and we recommend its use whenever practical. Both system and database administration skills will be required to implement it.
This Redpaper assumes that the project team works apart from the customer's IT team, generally in geographically distinct locations. Due to the geographic separation, the project team is assumed to have its own integrated development and test infrastructure, as well as processes. The customer IT team, on the other hand, may or may not have its own development and test infrastructure or processes.

Because this paper also assumes that most of the development work is to be performed by a project team, the project manager should not be overly concerned with synchronizing the project team's development and test environments and processes with those of the customer's IT team, if applicable, during the build phase. Due to the focus of this paper, most of the activities for the build phase is presented from the project team's perspective.

**Note:** It should not be inferred that working geographically separated from the customer is best practice per se. Working directly in contact with, as well as separate from, the customer has its pros and cons. Which approach to use depends very much on the individual wishes and preferences of both the project team and the customer.
4.3.1 The team development environment

Setting up a proper development environment is a critical project team activity, because the environment can be considered the “nerve-center” during site implementation. An efficient development environment will enable all team members, that is, the developers, to be fully utilized and productive when coding and unit testing site components. Conversely, an incomplete development environment can lead to frustration and significant productivity loss.

Figure 4-5 illustrates the different environments and how they are used during the various deployment stages.

Creating the development environment involves setting up a number of components. Typically, team development environments consist of a repository server to store Java and non-Java file assets, a shared database server, and individual code development systems. It may be necessary to have shared access to any external systems, such as an WebSphere Commerce Payments node or an WebSphere MQ server.

Figure 4-6 illustrates each element of the team development environment.
We will now describe each element of the team development environment. The names refer to Figure 4-6:

- **Database server.**
  Setting up a shared database for development and test purposes can help reduce setup time and should also reduce the chance for data errors. This database can reside on a testing server (as described in 4.3.2, “The shared test environment” on page 72), or on a dedicated database server machine, depending on the size of the project. If the project team consists of more than three to five developers, we would recommend setting this up on a dedicated server.

  Since the installation of WebSphere Commerce Developer V5.6 requires an installation of IBM DB2 Universal Database Enterprise Server Edition V8.1.5, using a shared server will not save hard drive space, but configured correctly, it can greatly reduce the processor and memory requirements, as the database server services can be disabled on the individual developer nodes.
In Figure 4-6, a dedicated database server hosts a number of databases, configured for code development, test, and QA activities. If the database stored on each developer's machine, it will be necessary to maintain and update each database whenever there is an update to either test data or database schema. During the build phase, the database tends to be updated or fixed regularly, and these updates will significantly reduce access time for developers if their systems must be updated. Further, application of a database-related fix or Fixpak must be duplicated on each developer's system as well. These are the main reasons for recommending a shared database server for a project team during the build phase.

- Source code management (SCM) server.

All projects, be they large engagements with many developers or small ones with only one developer, will benefit from a source code management system. The project team should implement one regardless of the size of the project.

- Developer clients.

The developers' client systems should be installed with WebSphere Commerce Developer V5.6. The developers' code development system should connect to the shared database and shared source code repository. All developers should code and unit test inside one of the WebSphere Commerce Developer integrated test environments (refer to 7.1, “Development tools” on page 116 for details about the development environment). After initial coding and testing has been completed, the project team should continue test activities in a project test environment, which is described later in 4.3.2, “The shared test environment” on page 72.

- WebSphere MQ Server.

Because most projects are likely to require some form connectivity to back-end systems, the project team should install and configure an WebSphere MQ server on either a separate system or using shared resources in the project test environment. An overview for situating

---

**Note:** Using a shared database may constrain the development environment so that constant connectivity is required. This would limit the scope for mobile development and may be inconvenient during circumstances such as presentations or demonstrations.

It is fairly simple to enable off-line support by utilizing the cataloging features in IBM DB2 Universal Database and allow developers to switch between a server and a local copy of the database for such scenarios, since the WebSphere Commerce Developer already contains a copy of IBM DB2 Universal Database Enterprise Server Edition V8.1.5.
WebSphere MQ in the test environment is described in 4.3.2, “The shared test environment” on page 72.

Where needed, WebSphere MQ client software should be installed and configured on developer systems.

- **WebSphere Commerce Payments Server.**

  Should the site accept online payments, the developers’ systems will need access to a shared WebSphere Commerce Payments system.

  As with a shared WebSphere MQ system, it is possible to install this software on a test server or use a dedicated server to host it. The developers requiring access to this functionality need to have their development system properly configured to use the WebSphere Commerce Payments.

  **Note:** WebSphere Commerce Developer V5.6 contains a WebSphere Commerce Payments test server. While it is possible to use this on the individual developer systems, it is recommended to use a shared system. As with the discussion about using a shared database server, there are situations in which it is preferable to use a local WebSphere Commerce Payments test server, for example when developing custom payment casettes.

Potentially, all the developers can be connected to the shared systems at the same time. The developer systems will host the development environment, IBM WebSphere Studio Application Developer, a database client, and other project-specific tools for developing non-Java assets.

The system administrator will play a key role in setting up the development environment. The database administrator will also be required to install and configure the shared database server. Refer to the next section for more details about team responsibilities.

**Team roles and responsibilities**

The details about the roles and responsibilities for the development team in setting up and configuring a shared development environment are provided here and summarized in Appendix A, “Project team creation and common roles” on page 225:

- **System administrator.**

  The system administrator must:

  - Install and configure the source code management system.
  - Install and configure a shared WebSphere Commerce Payments system, if needed.
– Install and configure a shared WebSphere MQ system, if needed.
– Document the process for connecting these various components and systems.

► Database administrator.
The database administrator must:
– Install and configure the database server.
– Create and populate a database for developers to use during the build and unit test phase.
– Create and configure a separate set of databases for WebSphere Commerce integration testing (as discussed in 4.5, “Subsystem integration test activity” on page 82), and Functional Verification testing (as discussed in 5.2, “Functional Verification Test (FVT)” on page 96).

► Developers.
The developers must install and configure:
– WebSphere Commerce Developer. This must be configured for using the team repository.
– Client software for source code management, if not included with WebSphere Commerce Developer.
– WebSphere MQ client.
– Other necessary software.

► Team repository administrator.
The team repository administrator must:
– Configure the team repository by importing the initial code versions to the repository.
– Import other required repositories.
– Create appropriate projects and packages as outlined in the macro design and micro design documents.
– Create users and assigns appropriate permissions, as outlined in the macro design and micro design documents.

For smaller projects, the system administrator can also act as a team repository administrator.

► Build manager.
The build manager is responsible for:
– Creating the build deployment process.
– Coordinating a build schedule with the team.
– Identifying all elements and the contents of each builds.
– Creating regular builds.
– Establishing roles and responsibilities of team members in creating and using builds.
– Establishing actions that must be taken when a build failure occurs.

4.3.2 The shared test environment

The project team's use of the shared test environment is described in detail in 4.5, “Subsystem integration test activity” on page 82 and 5.2, “Functional Verification Test (FVT)” on page 96, but it is helpful to highlight key points here.

The main objective of setting up a shared test environment is to integrate code pieces from all developers, in the form of unified code drivers, and deploy them on the test servers for comprehensive testing, as illustrated in Figure 4-5 on page 67.

One or more sets of test servers can be used for a project team. We recommend that each project team should have minimum of two test server configurations so that testing and code fixing tasks can be performed in parallel.

A test server topology can be single tier or multi-tier. The closer the match between topologies for the test server and the final production environment, the better the chances of finding and addressing any infrastructure or configuration related issues early in the test cycle.

If the test topology is in a single tier configuration, this means that the database, WebSphere Commerce, the WebSphere MQ server, and WebSphere Commerce Payments are installed and configured on a single hardware box. These test servers may also host shared facilities, such as the database, the WebSphere MQ server, or the WebSphere Commerce Payments software used by the developers during the build phase, as previously described. It is the sharing of physical resources that establishes the basis of the integrated development and test infrastructure for the project team.

4.3.3 Configuring a development process

In addition to setting up hardware and software, the project team should establish a development process that ensures consistent code development.
Doing so will create a good and effective team working environment. The development process should address, at a minimum, the following questions:

- Who should fulfill the role of team repository administrator? Although the system administrator might assume this role, this should be a deliberate rather than default decision.

- When should all team members receive new “drops” of the current code solution so that code written by the team can be integrated, used, and tested as the build phase progresses?

The development process must also take into account how database updates are to be managed as problems are discovered in the schema or either in sample or migrated data. The database administrator, with assistance from the data migration expert, would typically perform this role and would be responsible for coordinating appropriate changes to all instances of the databases on all servers with the system administrator.

### 4.3.4 Configuring a simulated production environment

The development and testing environment should mimic the production environment as closely as possible. However, there may be circumstances where it is not feasible to mimic fully the production environment due to hardware, software, and tool constraints.

In those situations, the project team may need to simulate a production environment in the development and test environments, especially when the development and test environments are not located at the client site. For example, if the project team is not at the client site, it may not be possible to access back-end systems, such as ERP systems, from the developer and test systems.

The back-end simulator should be designed to accept request messages and produce response messages, based on the request message identifier. Although the simulator may not fully test all back-end scenarios, it may be sufficient to test code that at the time requires back-end connectivity. This simulator could also be used for FVT, until the code is transferred to customer location for end-to-end system integration testing (as described in 5.3, “System Integration Test (SIT)” on page 98).

Following are two examples of how creating a simulated environment aided project teams:

- A customer was using a proprietary method of user authentication through a Lotus Domino server. In this case, the project team decided to simulate the authentication mechanism, using a JSP to pass in the authentication code.
This meant that during the build phase, the project team had a realistic way of testing this proprietary mechanism.

An alternative approach would be to install and configure a Lotus Domino server within the development and test environments. This alternative was rejected, because it would have required more effort to install, configure, and manage the server than creating and using the JSP-based simulation tool.

In a commerce engagement project, the team implemented a simulator that helped test the connectivity of WebSphere MQ with the customer’s mainframe systems. The simulator generated appropriate response messages for the test request messages. This simulator also helped test the format of the request and response messages.

The simulator designed by the services team also randomly selected a response message from a pool of test messages previously collected from real back-end transactions. The selection was constrained by the request message ID. This simulator helped the project team with troubleshooting connectivity, for example, by identifying message format mismatches, performance issues with back-end connectivity, and so on.

The cost of implementing a simulation tool such as this would obviously vary from customer to customer. This additional resource required should be determined early in the project, such that it can be incorporated in the project plan.

### 4.3.5 Source code management

As with any software development project, code components for an e-commerce site will be written by several developers working over a significant period of time. Any modifications to code need to be tracked and managed, so if a problem occurs, it is easier to identify the change or changes that caused the problem.

Using a source code management systems brings a number of benefits:

- It helps prevent problems when two or more developers makes changes to the same code component at the same time.

**Note:** Some SCM systems simply prevent simultaneous modification of the same file, while others, like CVS, has procedures to handle conflicts when two developers makes changes to the same parts of a file.

- Earlier versions of code can usually be obtained easily by backtracking through the code management system.

- Any change made to a component, along with the reason for the change and the name of the developer, can be listed.
4.3.6 The build process

The build process is a mechanism for creating a unified code driver that has all the necessary code and non-code pieces to allow some level of site testing.

The project team should create a build process for the project. This process could be shared with the customer IT team so that a similar activity can start at the appropriate phase in the project life cycle, usually after site launch. The build process should be finalized before exiting the design phase.

A build process may be created and used for code development during the build phase and, additionally, for all types of test phases. During test phases, the project team can use the build process to apply tests in order to build or generate test results.

**Note:** If the project requires developing the application on one platform (in this context, platform refers to any of hardware, operating system, database system, etc.) before deployment to a different platform, any differences in platforms must be incorporated into the Java code and JSPs as they are produced.

Despite the fact that these components are written in Java, there may be minor problems encountered in the porting that should be addressed as quickly as possible. For example, differences in the implementation of SQL may require conditional code in any developed session EJBs. Refer to 8.3.6, “Handling database vendor specific code” on page 151 for some tips to overcome these differences.

The following is a high-level description of the build and deployment process:

- The build schedule.
  
  The first task is to define a build schedule for both development and testing phases. During testing, the builds tends to be performed more frequently than during the development phase.

  It is important to start the build process as early as possible during development to establish how well the use case integration is progressing.

  We recommend creating a schedule that suits the project needs. Creating builds too often during implementation will likely require too much time to
deploy, which in turn, will prevent the development team from being able to use the builds.

Conversely, not creating enough builds during FVT may lead to a longer testing phase, as closing defects will take longer.

During the SVT and UAT activities, as described in Chapter 5, “Site test phase” on page 89, the build frequency should typically be once or twice a week. In practice, the frequency depends on the number of defects that get reported by the customer IT team.

Emergency builds can always be created during FVT and UAT activities when a build fails with critical, or "show-stopper", problem.

Roles and responsibilities.

Although creating a complete build involves participation from several team members, the project manager should assign one individual to be responsible for creating builds. This person may either be one of the developers or the team repository administrator.

The build person will be responsible for following the build process to create builds at the scheduled times. They should also ensure that code to be included in the build is delivered by the code-cut-off time. The code-cut-off time is when the developers decide to release their code as part of the packages to be included in the next build.

Depending on the SCM system being used, various procedures must be followed. With CVS, for example, it is common to tag the resources using a special naming scheme to mark files part of a given build.

Creating the build content.

Build creation involves a number of steps. Each build is composed of many components. In WebSphere Commerce V5.6, a build is packaged into an EAR file and deployed on a target WebSphere Commerce server.

If the specific environment requires additional deployment resources, for example database modification and population scripts, all of these should be packaged together into one ZIP or TAR file. Doing so will allow for storing a deployment as a single file. A deployment script that deploys all the resources in one action can then be devised.

Version control for builds can be managed more easily with one deployment file than with loose individual build files, especially if multiple deployment files are used for each build. It is usually possible to include the build files in a source code management system to get benefits from the version control mechanism.

A typical WebSphere Commerce V5.6 build includes:

– An EAR containing the JSPs, commands, EJBs and data beans.
Chapter 4. Build phase

4.3.7 Build automation

Automating the processes for build creation and deployment will save a lot of time. Most of the tasks can be automated through various scripts, with each task having a specific set of automated activities:

- A build can be created automatically by writing scripts that can extract assets from the SCM. Most SCM vendors provide tools to work with their products through scripting.
- Next, a script can be used to copy the build file over to the appropriate test servers.
After the file is copied over to the test server, a script on the test server can extract the files and copy them to the appropriate directories.

- DDL files may be used to update the database, if necessary.

- After the files have been deployed, the application must be restarted. This can be automated by using the `wsadmin` scripting facility, included with WebSphere Application Server V5.

- The `wsadmin` script, supplied with WebSphere Application Server V5, can also be used to deploy any new EJBs.

- Finally, we would recommend using the batch compiler script to precompile any JSPs.

**Tip:** A WebSphere Commerce V5.6 build and deployment tool is available from IBM. Refer to 7.3.2, “Auto-deploy tool” on page 142 for more information.

The complete build creation and deployment process can be performed through automated tasks, which are directed by scripts. The individual task scripts can be combined into a single larger script that can then be invoked automatically at build times.

Experience shows that build automation can save time, as well as reduce the risk of human error. Once the build is deployed successfully, the later build verification tests performed using testing tools (as described in Chapter 5, “Site test phase” on page 89) can also be automated.

### 4.3.8 Build failure

After the build has been successfully deployed, the project team should perform a Build Verification Test (BVT) on the resulting site to ensure the build is free from errors and that it has been applied correctly.

In most cases, the BVT will confirm the build status. However, in some projects, there will be occasions when the Build Verification Test fails. If a failure occurs, the build person must decide if a second build should be created immediately, or if the team should wait to create the build during the next build schedule.

The decision is likely to depend on many factors. If the build failed during the build phase, and the failure is not critical, for example, other components continue to function correctly, then it may not be necessary to fix the build and recreate it. However, if the build failed during a test phase, such as FVT, and the failure affects test cases being executed, then it maybe advisable to re-create a new build and redeploy it.
If the build failure cannot be remedied immediately, it may also be necessary to roll back the build by deploying a previous build to the test server.

### 4.4 The problem determination process

From time to time, the project team is likely to encounter problems. They may arise during the build phase, but will mostly surface during test phases.

Because a typical commerce site will consist of many components, such as WebSphere Commerce itself, WebSphere Commerce Payments, WebSphere MQ, a database, and directory services, any one or more of these components may fail or experience problems. The key to efficient problem resolution is first isolating the component where the problem surfaced and then identifying the root cause.

The recommended approach for isolating the problem is searching the log files. Each component will write to one or more log files.

**Note:** Each component may have to be configured for logging, and the component's documentation should explain how the component's parameters should be configured to start logging.

Most production systems are not likely to be configured for logging, because the task consumes valuable system resources and disk space. Normally, the default logging setting for production systems is to log only essential components. The logic is that the risk of missing the original cause of a problem must be balanced against the probability of a problem occurring, the resource demand, and the administrative cost of keeping excessive logging in continuous operation.

Problems can be categorized into a number of distinct groups. These are as follows:

- Installation or configuration.
- Build deployment.
- Runtime.
- Application defects.
- Performance.
- Others.
4.4.1 Component installation and configuration

The installation and configuration of many components also presents opportunities for problems to surface. If a problem were to occur soon after a component is installed, then the project team should review the component's installation logs for clues, as all components are likely to have log files. In addition, the project team should confirm (preferably reconfirm) that all prerequisite software has been installed, is at the required level, and that the necessary patches and fixes have been applied.

Check the product documentation for any troubleshooting tips to help understand installation problems.

Note: Any errors or problems encountered should never be dismissed as unimportant. With complex systems, the interactions between one or more “broken” components can often result in apparently unconnected errors or symptoms that can become increasingly difficult to diagnose by the project team as it progresses through the development process.

4.4.2 Build deployment

Problems may be encountered during build deployment and could be due to incorrect builds or errors resulting from the deployment process. The process of locating, diagnosing, and resolving problems with build deployment should begin by checking the log files.

Note: Unless the installation instructions or prerequisites explicitly state that later versions of a product may be used, you must ensure that you install exactly the correct version number for each component. Incorrectly using a more recent or later version for a component may invalidate the system configuration.

Similarly, before applying intermediate fixes or fix packs to WebSphere Commerce and its components, care must be taken to ensure that all prerequisite patch levels of supporting software are as required. Failure to do so may not be reported during the installation of the fix and a broken system may result from this.

Note: Some problems may surface as Java errors, and the Java stack trace information is not always very informative. The preferred manner for assessing the problem is reviewing the trace to determine which component may have caused the problem.
4.4.3 Runtime problems

The causes of runtime problems are many. The preferred starting place for tracing a runtime problem in the WebSphere Commerce log files. The log files will most likely point to the location of the failed component.

**Note:** For some runtime problems, you may want to see what system functions were running when the error occurred. Using a `tail` command to monitor the data recorded in log files in real time may pinpoint for you the event that triggered the error condition.

4.4.4 Application or customization defects

Problems or defects with application code are best debugged and fixed by using the development environment. Extra debug flags may be added into the Java or JSP code to help narrow the location of the problem. Specific checkpoint messages may be output from the code and will show up in the log files. These messages will provide line-by-line sequence of execution, which should help the developer to find and fix the problem.

**Tip:** The WebSphere Commerce Developer V5.6 environment can be configured with both a lightweight and a full test environment. Due to resource constraint, the full test environment should be run in debug mode. If the IBM WebSphere Studio Application Developer debugger is to be used, the development environment should be configured for the lightweight test environment. Refer to 7.1, “Development tools” on page 116 for more information about the test environments in WebSphere Commerce Developer V5.6.

4.4.5 Performance problems

Even though log files are the best place to find and isolate problems, their usefulness in diagnosing performance issues is limited, because not all performance issues are caused by component errors. If faced with performance issues, the project team should check system performance monitors first.

Most components ship with performance monitors to aid in measuring and tuning their performance. As with logging, however, the function is usually turned off. The project team should turn on (if currently off) appropriate component performance monitors and begin assessing the output. Please note that the performance monitors may adversely impact the existing performance, so the project team should plan their use accordingly. As with system logging, each
component's documentation should explain how to enable and disable its performance monitor.

4.5 Subsystem integration test activity

The final activities of the build phase involve using an integrated coding, unit, and integration testing method. After completing code build activities, the project team must integrate all the use cases to ensure that they function and communicate correctly together. The use cases must also integrate with the WebSphere Commerce subsystems, such as Member, Order, and Catalog. The project should start formal site testing only when all use case and subsystem integration tests have been finished.

Incorporating subsystem integration with the build phase is feasible; however, the project team is better served by keeping the activities separate. When the activities remain separate, the project team is capable of implementing code and then integrating it with greater efficiency than if performed through a combined approach. The main reason for the increased efficiency is that the project team can focus on related tasks, sequentially. The total gain will be affected by the structure and location of teams and subteams.

A checklist summary for the tasks and work products of the subsystem integration test activity is provided in Figure 4-7.

**Subsystem Integration Testing Activity**

<table>
<thead>
<tr>
<th>Input Work Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Build phase outputs</td>
</tr>
</tbody>
</table>

**Task:**
Code integration.

**Objective:**
Ensure proper code integration between use cases and WebSphere Commerce subsystems.

<table>
<thead>
<tr>
<th>Output Work Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Code integrated at all levels, which may include components, use cases and/or WebSphere Commerce subsystems</td>
</tr>
<tr>
<td>• Code that is ready for systematic/formal site testing within the specified development methodology</td>
</tr>
<tr>
<td>• Implementations ready for testing on the target WebSphere Commerce system</td>
</tr>
</tbody>
</table>

*Figure 4-7  The subsystem integration test activity.*
The project team will most likely not want to use the entire set of use cases for testing during this activity, as a subset would be more appropriate. The team should assess the use cases to select as small a set that will allow sufficient integration testing between the use cases and the WebSphere Commerce subsystems. Please note that there would be little value gained from testing outside the subsystem integration tests at this phase of the project, because to do so would only increase the complexity and number of tests that would otherwise need to be performed. All testing that is to be performed should be done on a common test server rather than in an integrated development environment running the test environment, because it will ensure that the integration tests are performed within a realistic environment. Using test cases also prepares the team for the more complex and comprehensive formal testing activities that are to come under the site testing phase.

The team that worked on the previous build phase may now be split, if needed, into test and problem fixing subteams. Please note that at the start of this phase, the site code, as a whole, was not ready for formal testing. Considering this fact, the project manager must ensure that the project team completes all levels of WebSphere Commerce integration before its team members are reassigned to other tasks.

### 4.6 Handling errors

Despite the integration testing, errors may continue to occur throughout the project and during normal operation of the site. For example, users may create errors after failing to supply a valid response in a form or on a Web page form.

Managing errors requires special attention to ensure prompt recovery. The project team will want to minimize adverse side effects, while providing as much detail to the team responsible for diagnosing and correcting the problem.

Each time an error or system exception occurs, the project team must assess the following six aspects of the problem:

- What will the user, for example a shopper using the site, see?
- What will the system administrators see when maintaining the site?
- What will the developers see, for example, when they develop, test, deploy, and troubleshoot the site?
- What will be the effect of the error or exception on the current transaction?
- What is the recovery strategy following the error or exception?
- After the error or exception, what state are the objects left in?
4.6.1 Handling each aspect in a real project

For a large project, IBM Software Services addressed the aspects of this particular project as follows.

The user aspect
As with most projects, the user interface was handled by JSPs. Each JSP was coded with error handling in mind.

There are three options for handling errors in this way:

- Each JSP can contain code to handle errors, where the code addresses all aspects of the error handling required.
- A JSP-specific error URL is registered with each JSP that determines what the user will see.
- A global error URL is registered for the whole store.

The system administrator
WebSphere Commerce V5.6 provides logging functions that support the issuing and logging of error messages. This function is fully documented in the product’s online documentation. Please note that the function is based on error codes, and it is enabled for multiple languages.

This facility was used to log the following:

- All exceptions and errors that occur.
- All suspicious situations, such as failure to authenticate a user.
- All severe problems.

Tip: For WebSphere Commerce V5.6 errors, detailed error description and resolution is available through the use of the WebSphere Application Server Log Analyzer. Refer to the WebSphere Application Server Information Center at the following address for more information on using the Log Analyzer:

http://publib.boulder.ibm.com/infocenter/wasinfo/index.jsp

The developers
The project used the WebSphere Commerce log facility:

- All errors.
- All warnings.
Refer to 8.5.3, “Command error handling” on page 174 for more information about handling error occurring in custom code.

The project used the WebSphere Commerce tracing facility:

- Default values used when no explicit parameter or value was specified.
- Success or failure of key operations.

**Note:** The failure would already be reported as an error, but reporting the higher level key operation failure is also important.

For example, the initial error could have been a failure to connect to a back-end ERP system to submit an order, and would be reported as a communication failure, while the reported key operation failure could be reported as the failure to process the order.

- Empty result sets for a given key or search criteria.
- Things not found or initialized.
- General messages about the state of the code, for example, while debugging.
- All assertions, where a developer would normally put a comment, such as:
  ```
  /*this should never happen*/, or /* why are we here*/
  ```

Refer to 8.5.4, “Tracing” on page 178 for more information about using the WebSphere Commerce tracing APIs in custom code.

**Recovery and object state**

Recovery, or the handling of an exception, means managing the effects of the exception to ensure they are corrected, and then restoring the normal flow of operation code.

Often one single component cannot deal with all the aspects of exception handling. Usually, it is a collaborative effort, where several components cooperate, with each of them handling different aspects of the exception.

**Important:** The WebSphere Commerce command framework performs transaction control and has the ability to retry commands that have been configured as retriable. These features must always be used for transaction control.

The WebSphere Commerce features for retrying a command are not always sufficient for recovering from a specific error condition. For example, on one project, a system resource could not be found. The component would try to
recover fully from the situation. First, it attempted to use a default name for the resource if a specific name had not been passed. Second, it would try to use a series of different or default locations to find the resource. The component would attempt to find the resource in the system class path and then the current directory. The component did not simply give up the first time the problem occurred. The result was a significant improvement in the robustness of the system, producing a site that was much more stable and resistant to error. The lesson from this is that a command must not just rely on letting the WebSphere Commerce framework retry the operation, if it is possible to implement alternatives in the control flow.

Not all exceptions signal an error condition. For example, when attempting to retrieve a non-existing row using CMP entity EJBs, the EJB container will throw the checked exception `ObjectNotFoundException`. The problem is to decide what should be done if the data required to proceed does not exist. In practice, the business logic components should resolve the issue. Depending on the context, it may be acceptable to not have data, or it may be an unacceptable and exceptional situation. If it is acceptable, the business logic should adjust the flow of execution to the appropriate scenario. If it is unacceptable, an exception must still be raised, again by action from the business logic components.

Note: Catching exceptions and doing nothing or simply logging the fact is not exception handling, it is exception masking. No component should mask exceptions, because it quickly becomes impossible to detect and resolve problems. Always recover from the exception, or handle as many aspects as possible and throw the remaining exception aspects again. Refer to 8.5.3, “Command error handling” on page 174 for techniques on signalling exceptions without necessarily throwing them again.

### 4.7 Summary

This chapter has discussed implementing code for WebSphere Commerce sites. Special attention has been paid to establishing an integrated development and test environment, because it is important to unit test site code as it is developed. Doing so enables the project team to uncover errors and correct them while code is still being developed. Automating the build process was highlighted to reinforce that fact that it can improve development efficiency by reducing time and decreasing errors. The discussion about problem determination emphasized the fact that problems should be resolved, whenever possible, as early in the development process as possible.
The chapter concluded with discussions about the importance of testing components as they are developed, as well as what the best practices are for developing code to correctly handle errors and exceptions from a production site.

In the next chapter, we discuss best practices for comprehensive site testing.
Site test phase

All the testing completed prior to the site testing phases described in this chapter have focused on integration of the site code at one of three levels: unit or component, use case, and WebSphere Commerce subsystem. The emphasis so far has been on ensuring that each of the customized components operate correctly within themselves.

For the site test phase we describe a series of activities that may be used systematically to test a newly created site or a migrated one prior to the “cutover” to production step. The emphasis of the testing is to ensure the customized site components interact correctly with each other, as well as with all relevant out-of-the-box components.

Note: The site test phase is not the only time during the project lifetime that testing activities are carried out. As already mentioned in Chapter 4, “Build phase” on page 59, testing activities are carried out throughout the build phase. Also static testing, such as design reviews, are also a constantly reoccurring activities. Refer to 3.6, “Design review” on page 56 for an example of the latter.
5.1 Site testing activities

Before entering the overall site testing phase, individual components should have been tested using Build and Verification Tests (BVT), as described in 4.4, “The problem determination process” on page 79. Once validated by BVT, acceptable components may be assembled and integrated and then tested as an integrated unit. The test activities discussed in this chapter are as follows:

- Functional Verification Test (FVT).
- System Integration Test (SIT).
- System Verification Test (SVT).
- User Acceptance Test (UAT).

As the topic implies, the site testing phase add value by ensuring that all code is tested at the site level before the site itself becomes fully operational. Unless otherwise indicated, all stress and performance tests should be performed on the actual site code.

Each activity focuses on a different aspect of the commerce site testing. Through experience, IBM Software Services has realized that the systematic approach outlined here helped streamline site testing during engagements, and as a result, improved their ability to deliver operational site code faster with a higher degree of reliability.

**Note:** Throughout the testing phases, it is essential that all test results and responses are recorded, along with all actions taken. This includes progress reporting, test pass or fail results, and problem determination procedures to assist with later problem resolution or troubleshooting.

An overview of the site testing phases is shown in Figure 5-1 on page 91.
5.1.1 Creating the site testing infrastructure and processes

Irrespective of who will perform the actual testing, the project team must ensure that an adequate test environment and appropriate test processes have been established and configured. Without a suitable test environment, namely one that reasonably resembles the target deployment system, all results produced using the test systems will be marginal at best, because they will not have been tested to operational standards. In addition, if test processes are omitted or must be developed at the last minute, the project will require time to validate test results, a resource demand that will surely extend the entire test process.

During engagements, IBM Software Services has identified the following activities as very helpful for configuring an adequate test environment and creating appropriate test processes:

- **Identify a test lead.**
  
  Both the customer project manager and project team manager should identify and nominate test leads. The customer and project test leads will assume responsibility for all test activity preparation and act as the liaison between the teams.

- **Prepare a simulated production environment for testing.**
This activity involves implementing a WebSphere Commerce test environment that has “production ready” characteristics, including simulations of all required back-end connections. As appropriate, the project team may adapt the test environment used during the build phase to jump-start site-level testing, such as FVT. The customer IT team should assume the responsibility for implementing the simulated test environment, because they will be best suited to ensure the system reasonably resembles the target production infrastructure.

► Prepare more than one test server.

At a minimum, the customer IT team should configure two test server systems. Having multiple test servers is especially beneficial when teams, such as a test subteam and a problem fixing team, share servers, because the various test servers can support simultaneous testing of regular builds. For example, one server could be used exclusively by the problem fixing team to correct builds, while another server would support the test team’s continued review of new builds. Configuring multiple servers will enable the various test teams to work in parallel assigned tasks to reduce the total time needed to test, correct, and validate system code.

► Ensure that a communication infrastructure is in place.

Another important task that should be resolved during this phase is that the project manager should secure a high-speed network connection to any needed back-end systems. This type of connection eliminates the need for a simulated back-end system, and more importantly, it will enable the test teams to test performance when communicating with the back-end systems. The project team will then be capable of resolving any communication issues prior to the customer IT team initiating the SIT, System, and UAT activities.

► Create testing, problem fixing, tracking, and reporting processes and mechanisms.

Establishing appropriate test processes may involve the use of off-the-shelf testing software or local tools and processes. The customer IT team is encouraged to explore all testing strategies, but they should base their decisions on the level of experience they have with available tools and processes. Selecting a strategy that maps experience with available tools and processes produces the best results in terms of time, effort, and investment. If testing tools are to be used during the site test activities, their acquisition and implementation must occur prior to the start of the site test. This responsibility rightfully belongs to the person acting as test lead.
It is important that the identified problems are visible to all team members and that the project manager is able to generate status reports to aid in progress tracking.

At a minimum, any test process should demonstrate the following characteristics:

- Test cases are assigned to a test team.
- Test cases are executed by the test team.
- Any problems or bugs are documented.
- The documented results are sent to the test lead.
- The test lead categorizes results according to criteria, such as component at fault and the severity of the problem.
- For each problem, the test lead assigns a developer to resolve the issue.
- The developer corrects the problem.
- The developer updates the test lead, advising that the problem has been corrected.
- The test lead should normally schedule a re-run of the test. There may be other component dependencies that must be met before a re-run can be planned.
- A test phase can be considered complete only when all test cases have been successfully executed and no problems have been found.

**Note:** The testing may uncover problems with code: defects. Information about these defects should be recorded to provide useful metrics that help assess and improve development and testing processes. Useful metrics include:

- Severity of defect.
- Average time to find defect.
- Method for correcting each defect.
- Actual time spent correcting each defect.
- Average time to fix defect.
- Number of errors caused by fix for original defect.

- Automate the BVT process.

Automating BVT enables test teams to streamline the validation of new builds while minimizing idle time related to waiting for test results. Off-the-shelf products can aid test teams to automate this phase, and they should be investigated to determine if they can be used during a project.

- Review BVT test cases.

The BVT test cases should be reviewed to ensure that all functionally critical paths are executed and tested.
5.1.2 Executing test cases

The two testing strategies that should be considered for a WebSphere Commerce project are the single-pass test strategy and the multi-pass test strategy.

An overview of the first test strategy, single-pass testing, is provided in Figure 5-2 on page 94.

**Note:** A useful guideline is that the execution of a BVT should not exceed 30 minutes. If it does, the BVT test case should be assessed to determine how it may be refined, subdivided, or updated to improve test efficiency.

The single-pass test strategy involves making only one pass through each test case. If a test case executes successfully, it will be marked complete. If, on the other hand, a test case fails either completely or partially, all remaining tests are halted and the problems or errors found get reported to the problem fixing team for immediate review and action.
Once the problem fixing team resolves a reported problem, the test case may be re-evaluated. The sequence of test → report → correct → retest should continue until the questionable test case executes without issues. Only then should it be passed as complete.

This testing strategy works well when there is limited interaction between use cases and test cases.

The test lead must review all test results so that they may ascertain the cause of problems. For example, a report may indicate that 10% of the test case was successful, but 90% of the test case could not be executed because of issues identified as one or more open problems. Being aware of all testing issues and progress, the test lead will then be able to report accurately the current testing status to the project manager.

An overview of the second test strategy, multi-pass testing, is provided in Figure 5-3.

![Multi-Pass Test Strategy](image)

**Input Work Products**
- Depends on specific test activities being performed

**Task:**
Execute test cases repeatedly.

**Objective:**
Repetitively test code to ensure it is free from error.

**Output Work Products**
- Depends on specific test activities being performed

**Key Influencers:**
- Substantial interaction between use cases and test cases
- Time available to complete test activities
- Testing resources are available to oversee the tests

**Project Type:** Either new or migration  **Project Phases:** Site Testing Phase

*Figure 5-3  Multi-pass test strategy.*

The multi-pass strategy involves making multiple passes over the suite of test cases. Normally, at least two passes will be performed. The first pass should
follow the same process as that detailed for the single-pass strategy illustrated in Figure 5-2 on page 94.

Once the first pass has been run, the test lead must select a subset or the complete set of test cases for the test team to repeat the testing process from start to finish. The second pass, or subsequent ones, should take less time to complete than the first, because any issues found would be fixed prior to the start of the subsequent passes.

This testing strategy works well when there is a considerable interrelationship between use cases and test cases.

5.2 Functional Verification Test (FVT)

The main objective of the Functional Verification Test is to test the functionality of the site code from end to end, using a suite of test cases. The project team generally performs FVT activities before passing the code to the customer IT team for further testing. This phase is especially important when the project team is based at a different location than the customer test site or using a different environment than the one used during SIT, System, and UAT activities. If the project team is indeed at a different location, the FVT environment should mimic, as closely as possible, the production environment so that any back-end integration issues may be resolved as early as possible.

If, on the other hand, the project team is based at the customer location and has access to the simulated production environment, the FVT and SVT activities, described in 5.4, “System Verification Test (SVT)” on page 101, can be combined.

If FVT and SVT activities are combined, we recommend using the multi-pass testing strategy described in Figure 5-3 to ensure that all aspects of the site code get fully tested and are shown to be free of bugs or other issues.

An overview of the FVT activities and work products is provided in Figure 5-4.
Figure 5-4  Functional Verification Test (FVT).

It is important to reassess the activities listed in 5.1.1, “Creating the site testing infrastructure and processes” on page 91. Most of the activities should be completed prior to beginning FVT.

5.2.1 Team organization during testing

Testing and problem fixing may either be performed by separate teams or by a single team, preferably the one that participated in the build phase of the project. There are two possible strategies to consider to determine how best to organize your test teams:

1. The original build team is divided into testing and problem fixing subteams. The test lead can then assign test cases to the teams. The main advantage of this strategy is that it is not necessary to add new resources to the project; each test member was originally a developer on the project and, therefore, will possess a deeper understanding of the code and will be better able to test or fix it. However, for this strategy to be valid, it is essential that test cases are assigned to individuals who did not develop the functionality being tested.

2. A completely separate and independent set of testers is identified and assigned to perform the FVT. This group has the sole mandate of performing the FVT. The advantages of this strategy are a quick release of the original development team and a clear and fresh perspective on the code being
typically, problem fixing takes less effort than the original implementation, even allowing for the fact that the new testing team may not be familiar with the original design and development work. However, this approach is only possible if the previous phases were executed successfully.

Irrespective of which strategy is adopted, the FVT problem fixing and testing subteams should collectively decide and agree on a build frequency that supports the objective of completing the FVT within the project schedule. Before new builds are put on the test servers, code should be run through the Build Verification Test (BVT) activity, preferably using automated scripts, to ensure no issues exist.

FVT progress should be reviewed regularly, preferably on a daily basis. The review would normally involve:

- Monitoring a predefined percentage of test cases to see how many have been completed.
- Tracking the number of remaining open problems reported.

The FVT team members who performed the test cases and fixed the reported problems should be responsible for reporting test results to the test lead.

The FVT phase should also include executing stress test scripts once the code approaches a stable state. Executing stress test scripts should continue throughout FVT to ensure that problems, such as memory leaks, are caught and fixed before the FVT completes. Problems discovered during stress testing need to be reported to the development or problem fixing team, because these issues will need to be fixed and retested during FVT. Tools to automate the running of scripts should be considered, because they will free resources to concentrate on other test or repair activities. If tools are used, the system administrator will assume responsibility for implementation.

### 5.3 System Integration Test (SIT)

The customer IT test team needs to plan, prepare, and execute System Tests and UAT using a simulated production environment. The closer the test environment is to the actual or planned production system, the easier it will be for the team to launch the site after testing.

The SIT activity is a preparatory phase that determines if the customer test environment is ready to support System Tests and UAT. An overview of the activities and work products of the System Integration Test (SIT) phase is provided in Figure 5-5.
Unlike previous test activities, the SIT should be conducted by the customer IT team. It is important to make decisions up front as to the scope of testing and the nature of test infrastructure used for the System Tests and UAT. Ideally, the infrastructure should map as closely as possible to the planned production environment. If achieving this close approximation is not feasible, careful planning will be required to ensure that the System Tests and UAT will identify all issues with regard to the production environment.

The customer IT team needs to consider a number of aspects when preparing a test environment. It is essential that these aspects are addressed prior to commencing the System Verification Test, and it is advisable to begin these tasks while the FVT is underway. In many cases, intermediate code that has not yet gone through complete FVT can still be used to support the following main tasks:

- Select and implement an appropriate hardware-software configuration for the test environment.

You may want to use either a single-tier or multi-tier hardware configuration that supports either horizontal or vertical scaling through software configuration. No matter how you configure the test environment, you must ensure the final configuration is suitable for System Tests and UAT. Your system configuration decisions should be based upon capacity planning requirements.
If replicating the production environment proves to be infeasible, you should still refer to capacity planning information when configuring your test environment, because it will enable you to develop production style scenarios.

- Install WebSphere Commerce and the associated software stack on the selected hardware and software configuration.

If a scaling architecture has been selected for the production environment, ensure that the corresponding test environment has the required setup.

- Configure WebSphere Commerce and the other software components, such as WebSphere MQ servers.

The test environment should be configured accordingly with appropriate software that will enable it to communicate properly with back-end and legacy systems. If the customer IT infrastructure is maintained by multiple teams, the test leads must work with all the teams to ensure the appropriate infrastructure gets configured for test.

- Apply and test code developed by the project team.

This task is necessary to ensure that all back-end communication through any customized code works according to the design. For example, messages from WebSphere Commerce must travel successfully to back-end systems, and responses must be properly picked up by the project specific code.

This task is an important milestone for both the project and customer IT teams, because it determines when SVT should begin. Completing it may take time, especially when different customer IT teams own different parts of the communication infrastructure, so it is advisable to carry out some advanced planning.

In order to successfully achieve this part of SIT, it will be necessary to understand the code deployment process for the test servers. For BVT activities, the project team can transfer the knowledge, and in some instances, share BVT scripts with the customer IT team to automate this task. For code deployment, where possible, the project team should automate this task through the use of scripting language, such as Perl.

- Use a subset of test cases that supports test drivers.

This process is sometimes referred to as a “happy-path” test case. Happy-path testing should focus on testing the integration points between the commerce system and any back-end or legacy systems that the customer test environment connects to.

- Report all problems identified in this activity back to the project team.

The customer IT test team should establish simple processes for the following:

- Tracking of problems discovered during SIT.
– Reporting the problems to the project team.
– Retesting fixes.

This process of testing happy-paths and reporting problems should be repeated as often as necessary until such time as the test environment has been validated as acceptable.

5.4 System Verification Test (SVT)

The System Verification Test is typically conducted by the customer IT team, and their objective is to validate the functionality of site code received from the project team. All system tests should be performed in the simulated environment. An overview of the activities and work products of the System Verification Test is provided in Figure 5-6.

![System Verification Test (SVT)](image)

**System Verification Test (SVT)**

**Input Work Products**
- FVT activity outputs
- SIT activity outputs
- Stress and performance test scripts
- Test cases

**Tasks:**
Test target code on simulated environment that reasonably resembles the production system and its connections. Perform stress tests.

**Objective:**
Validate code function to ensure it maps to site design and performs as expected under load.

**Output Work Products**
- Site code ready for UAT

*Figure 5-6  System Verification Test (SVT).*

The customer’s IT team should undertake more testing responsibilities during the System Verification Test, while the development teams continue to focus resources on problem resolution. The customer and development test leads
should employ agreed upon processes as defined in their joint test plan. Mechanisms for reporting and fixing problems should be well established and used throughout SVT.

To validate the appropriate SVT infrastructure is in place, the customer IT team is encouraged to review the activities listed in 5.3, “System Integration Test (SIT)” on page 98. Reviewing this section will also enable the customer IT team to validate the testing strategy.

It is during the SVT activity that the site code first communicates with external systems. These systems may either be live or test ones, and they may consist of back-end systems, or specific ones, such as tax systems, data load, or feed programs. To validate the test environment, the customer IT team should conduct appropriate stress and performance tests, because the closer the environment can be made to approximate the target production system, the more reliable and useful the test results will be. The project team can share its stress test scripts with the customer IT test teams if both teams use the same tools to measure site stress test results. Capacity planning documentation can be an effective tool for assessing potential performance.

Any problems or unsatisfied results uncovered during either stress or performance testing should be discussed with and worked on by the project team. In practice, it is often the case that custom code developed by the project team is rarely the sole cause for a target site’s performance issues. A variety of actions must be taken to ensure performance targets are achieved, such as tuning the test hardware and software.

A successful SVT is likely to include not only successful function tests, but successful stress and performance tests as well. However, performance targets can be less than desired for the live site if the test environment failed to mirror the production environment.

5.5 User Acceptance Test (UAT)

The User Acceptance Test focuses on the final site look-and-feel, the site flows and behaviors, and overall business process validation. The customer's business team generally is responsible for validating UAT tasks. An overview of the tasks and work products of the User Acceptance Test (UAT) activity is provided in Figure 5-7.
Chapter 5. Site test phase

The UAT is conducted on the same test environment that was created for the SVT. The project team in this activity can provide the same support as it did during the SVT, specifically problem resolution. If all of the previous test activities were successful, then UAT is likely to be simple, because the business team will only have to validate the look-and-feel and business operations of the target site.

The customer IT team can act as the communications link between the customer and the project team, through which problems can be reported, tracked, and when fixed, re-deployed.

If the customer's IT team has been given the responsibility of testing the look-and-feel and business functionality of the site, then the SVT and UAT activities may be combined. If SVT and UAT are combined, we recommend the test objectives be fulfilled prior to site launch.
5.6 Summary

In this chapter, we identified and described the major testing activities for a commerce site. We discussed creating and applying a test infrastructure, as well as integrating it with back-end and legacy systems.

A key point throughout the chapter was that System Tests and UAT extend testing beyond the component tests conducted during the build phase. Also, the customer teams play a significant role in leading and conducting the tests, and as a result, they are properly positioned to assess more effectively what gets delivered against that which was proposed and required.

After successful conclusion of system testing, the customer will be ready to implement the proven, but non-operational code. The task of setting up the site to go live is discussed in the next chapter.
Launch phase

To this point, the design, build, and testing phases for the new commerce site take place within a clean and controlled environment. Each activity is planned and prepared in advance. Each component is used either in isolation from other components, or as part of a very controlled interaction.

In a live site, each of the components will be subjected to real user tasks. Real data is used to support the site functionality, and all transactional data and status information becomes mission critical. It is only when the actual component code is deployed to the production environment that real performance can be observed and fine-tuned.

In most transition projects, the new site will run in parallel with the old site to ensure a successful cross-over of functionality. This may introduce complexities of duplicate data sets that must be carefully managed and later synchronized.

In this chapter, we describe the final activities involved in preparing a production site for launch.
6.1 Site deployment and launch activities

All of the project phases so far have focused on preparation of a commerce site that is fully functional and well tested. However, there are still certain activities, some of which started during the design phase described in Chapter 3, “Design phase” on page 27, that must be completed before the production site can be made available to end users.

The main objective of this phase (as shown in Figure 6-1) is to ensure that the production environment can support the new or migrated code, with the expected capacity and performance.

An overview of the activities and work products of the launch phase is provided in Figure 6-2.
If the production environment is separate from the SVT and UAT environments, then all the steps listed in 5.3, “System Integration Test (SIT)” on page 98 need to be repeated to prepare the hardware and software environment for launch. These activities include:

- Setting up hardware.
- Installing software.
- Configuring the hardware and software.
- Connecting the resulting production-ready system to the back-end or legacy systems, as needed.
- Applying UAT validated code builds.

A capacity planning activity should be performed that reviews site traffic and load estimations and confirms that the hardware and software architecture for the production environment is sufficient to handle the anticipated capacity. The capacity planning activity may start as early as the design phase described in Chapter 3, “Design phase” on page 27.
Similarly, site infrastructure activity, as well as testing of new or migrated code, should be finished and refined, as needed, prior to site launch to ensure that targeted capacity requirements can be achieved at launch.

Some basic site hardware and software tuning may be required to improve site performance. The tuning may take place just prior to going live once all components are installed and operational within the production environment. Subsequent tuning may also be required as performance is observed and assessed over time.

System tuning may affect the database, WebSphere Application Server, or WebSphere Commerce itself. It may be helpful to run stress and performance scripts to observe actual performance, rather than to look for problems. Doing so will enable the team to gauge what tuning may be required. Any tuning steps performed during the earlier SVT and UAT phases should be repeated on the production environment to confirm the validity of the results and any changes made.

Performance and stress tests conducted during SVT provide a means for comparing the actual production configuration of software and hardware and the customized code against desired capacity targets. If after conducting performance and stress tests, and tuning the code, the production site fails to perform properly, the customer IT team should consider the following actions prior to launch:

- Tuning the site, focusing on the software stack or the hardware.
  
  The tuning might include making changes to the lower-level software components, that is, those components closer to the operating system level than to the WebSphere Commerce level. For example, the team might modify buffer size, caching configuration parameters, and network connectivity time-out values.

- Making changes to the production architecture of software or hardware.
  
  In extreme cases, the team may need to upgrade the architectural components of the production system. For example, the team might increase memory within the servers, or add faster processors.

This last option should be pursued only if all prior actions fail to produce desired improvements. Please note that proper capacity planning early in the project should minimize the likelihood of ever having to pursue this last option.

For existing sites that are being migrated, down time is a probable characteristic of a production launch. In order to assess the expected down time, the customer IT team should conduct partial or full mock launches. This will enable the team to identify all the steps required to bring up a new site.
After this data is available, it is sensible to compare it with the forecasted down time that the business can accommodate. This assessment will aid the customer teams in determining appropriate actions for reducing the launch window, which may be measured in hours, to a manageable limit.

For migration projects, another concern is the possibility of overrun in the down time while a new site is brought online. It is advisable for the team to consider and plan for the possibility of launch failures or other unexpected issues that may arise during the launch window. The following considerations will enable the customer team to assess launch contingencies:

► When the next launch is scheduled.
► How the next launch is scheduled.
► How to back up and restore the old site until the newly migrated site becomes error free.

Contingency planning will lessen the impact associated with failures in hardware, software, or networking components of the system.

Decisions made during this phase of a project depend on many factors, such as site complexity, availability, and the number and type of business transactions. Because many of the decisions affect more than just the site infrastructure, the customer IT teams need to play a lead role in overseeing and directing site launch preparations. Intimate knowledge of operational details and practical experience will enable customer IT teams to prepare the commerce site efficiently for operational availability. The development teams should transfer site knowledge to the customer IT teams to prepare each for post-launch activities, such as site maintenance.

The project team can also use this phase to transfer site knowledge to the customer IT support teams so that they can support the maintenance of the site after launch.
6.2 Summary

This Redpaper introduced a project development method that was specifically adapted from industry standards for the creation of e-commerce sites using the WebSphere Commerce family of products. Given that a typical development method consists of six phases, we made a conscious effort to focus only on the following four core development phases:

- Design
- Build
- Site test
- Launch

The reason is simple: When developing e-commerce sites using WebSphere Commerce, it is critical for the project manager, as well as the team, to understand:

- The key input work products for each phase.
- The core tasks that must be accomplished to complete each phase.
- The skill requirements needed to support the tasks.
- The output work products to deliver during the phases.

The project development method outlines a structured approach that enables teams to build on the work products produced during the preceding phase. This “incremental” approach significantly reduces project risk, because it enables project teams to identify design, code and integration issues in a timely manner throughout the project's life cycle, so if problems arise, each may be corrected with minimal interruption to the project's schedule.

As specified in this Redpaper, the design phase marks the beginning of a typical WebSphere Commerce project, and it consists of three distinct activities:

- Solution outline.
- Macro design.
- Micro design.

The solution outline provides input to macro design activity, which in turn, provides input to the micro design activity. Following this incremental process, the project team will identify skill requirements, outline a project plan, establish a project team, and then define the project's functional requirements and document the detailed design decisions that are required to implement the target site. Actual execution of the phases will depend largely on the intricacies of the site, and the project team is free to adapt and employ any strategy, so long as the
strategy used is valid and consistent with the concepts presented in this Redpaper.

Completion of all design activities enables the project team to initiate the implementation phase. Ideally, the project team should pursue the continuous and incremental development and code integration testing strategy during this phase. This strategy advocates anchoring development on use cases and then testing code prior to it being integrated into builds. In essence, the project team codes site components, tests each, and then integrates the tested components into builds. Because the project team will be managing the integration of builds, the project manager will want to ensure the team employs an appropriate build verification test process, as well as an integration test strategy. The project manager should identify key people on the team to oversee the efforts, as well as to establish schedules, assign responsibilities, and create failure and error-handling processes. After the site components have been fully integrated, the team may start site testing.

During site testing, the project team should employ a systematic test strategy. The objective of the test strategy is to ensure the site components interact correctly and that each works well with out-of-the-box WebSphere Commerce components. The key site testing phases are as follows:

- Functional Verification Test (FVT).
- System Integration Test (SIT).
- System Verification Test (SVT).
- User Acceptance Test (UAT).

Employing a systematic test strategy streamlines the site testing process, which results in the project team being able to deliver reliable operational site code more quickly. The project team should create a suitable test infrastructure, one that resembles the production system closely, and define test processes to support the test strategy properly. Doing so will ensure the test results are meaningful and that they can be used to improve code reliability. How the project team is structured to support testing depends on the project's requirements and availability of skills, but the project manager is encouraged to structure resources so that there is a clear division of responsibility between test resources and problem fixing resources.

The FVT phase requires the test team to function test site code from end-to-end using a suite of test cases. SIT is an interim phase that requires the test team to assess the test systems to ensure each can support SVT and UAT. During SVT, which the customer test team should undertake, the site code is validated to ensure it operates as designed. Doing so prepares the customer team to conduct UAT, which is the testing of site look-and-feel and behavior. The key reason for employing a systematic test strategy is to ensure all code operates as designed,
and if any bugs are found, that they can be passed to the problem fixing team for prompt resolution.

The final project phase, as outlined in this Redpaper associated to the core development phases, is the launch phase. During this phase, the project team must assist the customer to assess and ready the production environment to support the new or newly migrated site code. The key activities include:

- Hardware preparation.
- Prerequisite software preparation.
- Hardware and software configuration.
- Code implementation.

In addition to the standard deployment activities listed, the project team should also encourage the customer team to plan contingency activities to mitigate potential issues or failures that may arise after launch. Contingency planning is prudent, and it can be as simple as identifying how the old site can and should be restored if the new site fails.

To conclude, the purpose of this Redpaper is to present a viable development method that a WebSphere Commerce development team, whether comprised of IBM consultants, third-party consultants, or customer team members, can use to create or migrate WebSphere Commerce sites. Contained within the body of this paper are numerous recommendations, examples, and notes to help the reader understand and then employ the method and relevant strategies. In addition, it provides extensive coverage on how tools that can be used to migrate specific site assets, and in some cases, even how it may aid in the deployment of new ones. When used properly along with a systematic development method, tools can significantly improve the efficiency of a project team to create or migrate WebSphere Commerce assets reliably.

The following sections in this paper provide details on the best practices to create or migrate WebSphere Commerce sites.
Development and Migration Best Practices
Development and migration tools

This chapter provides details about the available tools for development of new WebSphere Commerce sites and for migrating sites from older versions of WebSphere Commerce to WebSphere Commerce V5.6.

In addition to the tools contained in the WebSphere Commerce V5.6 distribution, we also discuss a number of additional tools that can be an aid in the development and migration of WebSphere Commerce sites.

The chapter is organized into the following sections:

- Development tools
- Migration tools
- Additional tools
7.1 Development tools

WebSphere Commerce Developer V5.6 is based on Eclipse technology. The development environment is a fully configured workspace for WebSphere Studio Application Developer with various enhancements:

- Unlike the VisualAge for Java-based development environment for WebSphere Commerce Suite V5.1, a WebSphere Commerce run-time environment is not required.
- WebSphere Commerce provides two pre-configured servers for your testing needs in development environment: Full WebSphere Commerce Test Environment and Lightweight WebSphere Commerce Test Environment.
- The Full WebSphere Commerce Test Environment uses the full WebSphere Application Server Base Edition to give you a more precise testing experience.

The following sections will describe the main differences between the two types of test environment and then go on to discuss other tools contained in WebSphere Commerce Developer V5.6.

7.1.1 Lightweight WebSphere Commerce test environment

The lightweight test environment is the default test environment in WebSphere Commerce Developer V5.6. The test environment can be configured to use IBM Cloudscape, IBM DB2 Universal Database V8.1.5, as well as Oracle 9i for the underlying database.

**Limitation:** WebSphere Commerce Developer Express V5.6 does not support Oracle 9i for the development database. Only IBM DB2 Universal Database V8.1.5 and IBM Cloudscape are supported by WebSphere Commerce Developer Express V5.6.

As the name implies, the lightweight test environment has a smaller resource footprint than the full test environment.

Due to the resource requirements of the full test environment, we recommend that you use the lightweight test environment, unless your development needs can only be fulfilled by the full test environment. Refer to the next section for details about choosing the test environment.
7.1.2 Full WebSphere Commerce test environment

The full test environment is closer to the actual WebSphere Commerce V5.6 runtime, running inside the WebSphere Application Server V5 test environment, supplied with WebSphere Studio Application Developer.

**Limitation:** The full WebSphere Commerce test environment does not support the use of IBM Cloudscape for the development database.

Refer to Table 7-1 on page 118 for detailed information about database support across the various WebSphere Commerce editions and test environments.

Like the WebSphere Commerce runtime, the full test environment does not support dynamic reloading of classes and JSPs. As such, development of databeans, commands and JSPs can be very tedious in the full test environment. Also, running the server in debug-mode is not practical due to the high resource requirements of the full test environment.

You should only use full test environment if any of the following is true:

- Your application requires additional data sources, beyond those used by WebSphere Commerce.
- You are developing against the WebSphere Application Server DynaCache API.
- You are developing code that requires WebSphere Application Server specific extensions to the servlet container.
- You are developing stateful session beans.
- You are working on performance issues.

If none of these are true, you should use the lightweight test environment.

**Note:** Not all developers have to use the same type test environment at the same time. This allows for dividing the work between the developers to minimize the number of developers that have to use the full test environment. Also, since it is easy to change between the lightweight and the full test environment, the developers can switch between environment, based on the test environment requirements for their current task.

7.1.3 Comparison between the test environments

Table 7-1 is a matrix of supported databases in the two possible test environments on the various WebSphere Commerce editions.
Table 7-1  Database support for the various test environments and releases

<table>
<thead>
<tr>
<th>Edition</th>
<th>Lightweight Test Environment</th>
<th>Full Test Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business</td>
<td>IBM DB2 Universal Database Enterprise Server Edition V8.1.5</td>
<td>IBM DB2 Universal Database Enterprise Server Edition V8.1.5</td>
</tr>
<tr>
<td></td>
<td>IBM Cloudscape</td>
<td>IBM Cloudscape</td>
</tr>
<tr>
<td></td>
<td>Oracle 9i</td>
<td>Oracle 9i</td>
</tr>
<tr>
<td>Professional</td>
<td>IBM DB2 Universal Database Enterprise Server Edition V8.1.5</td>
<td>IBM DB2 Universal Database Enterprise Server Edition V8.1.5</td>
</tr>
<tr>
<td></td>
<td>IBM Cloudscape</td>
<td>IBM Cloudscape</td>
</tr>
<tr>
<td></td>
<td>Oracle 9i</td>
<td>Oracle 9i</td>
</tr>
<tr>
<td>Express</td>
<td>IBM DB2 Universal Database Express Edition V8.1.5</td>
<td>IBM DB2 Universal Database Express Edition V8.1.5</td>
</tr>
<tr>
<td></td>
<td>IBM Cloudscape</td>
<td>IBM Cloudscape</td>
</tr>
</tbody>
</table>

Table 7-2 shows a comparison of average startup times for the test server using different test environments and database products. The tests were carried out using WebSphere Commerce Developer Business Edition V5.6 with no fix packs applied and not performance tuning on any of the databases. The actual startup times for your environment will vary from these, but the timings should give you an idea of the relative differences in response times for the different environments.

Table 7-2  Timing comparison between test environments and databases

<table>
<thead>
<tr>
<th>Database</th>
<th>Lightweight Test Environment</th>
<th>Full Test Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM DB2 Universal Database</td>
<td>1:58</td>
<td>3:29</td>
</tr>
<tr>
<td>IBM Cloudscape</td>
<td>1:12</td>
<td>n/a</td>
</tr>
</tbody>
</table>

7.1.4 Command line utilities

The WebSphere Commerce V5.6 has a long range of command line utilities. These are useful for development and daily site maintenance. In the following sections we will describe the main features of the following command line utilities:

- Loader
- ID resolver
- Text transformer
- XML transformer
- Extractor
- JSP batch compiler
- Encryption tools
Database cleanup utility

Loader
The `load`, or `massload`, command is a cross-platform utility for importing data into a database. The tool can load data into any of the following database providers:

- IBM DB2 Universal Database
- IBM Cloudscape
- Oracle 9i

The source format is flat XML file where each XML element represents a row in the database and the attributes for the element contains the data for the columns of that row.

The main advantage of using the loader tool, as opposed to executing files with SQL, or importing comma-delimited files, is platform independence. The same file can be loaded on any of the platforms and database systems, supported by WebSphere Commerce. This gives great freedom in the development process. For example, the development team can develop the site using IBM Cloudscape or IBM DB2 Universal Database under Windows and then deploy the solution to a WebSphere Commerce runtime running with Oracle 9i on Linux, without having to create platform-dependent import scripts. In this example, it is obviously important that the development team uses as database vendor neutral coding as possible in case they need to develop SQL code. Refer to 8.3.6, “Handling database vendor specific code” on page 151 for information about this.

Another advantage is that, in combination with the ID resolver (see “ID resolver” on page 120), there is no need to hard-code generated primary keys in the SQL, and the import files can thus be used on any system, without imposing restrictions upon the existing data with respect to such generated primary keys.

Example 7-1 shows an example of an XML document containing a catalog stub.

Example 7-1 Sample XML document for loading a catalog stub

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE import SYSTEM "mystore.dtd">
<import>

  <catalog
```
ID resolver

The ID resolver is a utility for transforming an XML document containing symbolic references, such as the one shown in Example 7-2, into an XML document containing static identifiers, such as the one shown in Example 7-1 on page 119.

Example 7-2   XML document for catalog stub with symbolic identifiers

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE import SYSTEM "mystore.dtd">
<import>

<catalog>
  catalog_id="$catalog_master"
  member_id="$MEMBER_ID;"
</catalog>

</import>
```

**Limitation:** The XML document in Example 7-1 contains the IBM DB2 Universal Database specific syntax for the current time, CURRENT_TIMESTAMP. The document is thus not platform independent. In order to work with Oracle 9i, the text would have to be changed to NOW.

The massloader unfortunately does not currently provide a platform independent method of loading time stamps.
The differences between the XML document shown in Example 7-2 and that of Example 7-1 have been emphasized with bold typeface. Example 7-2 uses four features to avoid literal identifiers in the XML document:

- **XML entities**
- **Automatic ID resolution**
- **Internal symbolic ID references**
- **External symbolic ID references**

These techniques are described in the following.

**XML entities**

Since the ID resolver uses a standard XML parser to read the source document, it is possible to utilize XML entities for identifiers that are of a more static nature but still may change, such as the \&MEMBER\_ID; entity in Example 7-2, or for identifiers that are really static, but that you do not wish to keep literal in the file, such as the \&en\_US; entity.
Note: Due to the fact that XML entities are expanded during ID resolving, it is necessary to escape such references if there is a requirement to store these literal in the database.

For example, the site requirements may be such that a given category must have the following value in order to be output correctly in the resulting XML:

\[
B\&oslash;ger
\]

If the input XML is first run through the ID resolver and then the massloader, the XML must contain the following value:

\[
B\&\&oslash;ger
\]

Note that the ampersand has been escaped twice. This line will look as follows after being run through the ID resolver:

\[
B\&oslash;ger
\]

When read by the massloader’s XML parser, the massloader will then store the following value in the database:

\[
B\&oslash;ger
\]

These entities must be specified in the DTD file for the massloaded file. In the case of Example 7-2, the DTD would look similar to Example 7-3.

Example 7-3  Sample mystore.dtd file

```xml
<!ELEMENT import (( catalog | storecat | catalogdsc )*)>
<!ENTITY MEMBER_ID "-2001">
<!ENTITY en_US "-1">

<!ELEMENT catalog EMPTY>
<!ATTLIST catalog
    catalog_id CDATA  #REQUIRED
    member_id CDATA  #REQUIRED
    identifier CDATA  #REQUIRED
    description CDATA  #IMPLIED
    tpclevel CDATA  #IMPLIED
    >

<!ELEMENT catalogdsc EMPTY>
<!ATTLIST catalogdsc
    catalog_id CDATA  #REQUIRED
    language_id CDATA  #REQUIRED
    name CDATA  #REQUIRED
    shortdescription CDATA  #IMPLIED
```
Automatic ID resolution

When the ID resolver encounters an XML element in the input document that corresponds to a database table with an entry in the KEYS table, and the XML element does not contain a valid ID for that column, the ID resolver will try to deduce the correct ID. If the ID cannot be deduced, the ID resolver will assign a new generated ID for that element. The ID resolver will also update the KEYS table accordingly.

The ID resolver will use the unique indexes defined for the table to determine the ID. For the XML document in Example 7-2 on page 120, when the ID resolver encounters the catalog element, it will

- Look up the table name in the KEYS table.
- If the table was found, check the attribute, corresponding to the COLUMNNAME column in the KEYS table.
- If such an attribute exists with a valid ID then stop and use that value.
- Check the contents of the target table for rows matching the current element, using the unique indexes for the table.
- If a row exists, matching the current element, assign the ID of that row to the current element.
- Otherwise, create a new ID for the element and update the KEYS table accordingly.

Internal symbolic ID references

No matter if the ID is generated or deduced from the context, if an attribute for the ID column is specified using the following format, the ID resolver will then assign the deduced or generated ID to that variable:

columnname="@reference"

In the following, other elements can refer to that value using the same syntax:

columnreference="@reference"

In the XML document in Example 7-2 on page 120, the following attribute specification occurs three times:

catalog_id="@catalog_master"
The first time, the attribute is the primary key for the CATALOG table. Since the expression @catalog_master is not a valid ID, the ID resolver will assign the actual catalog ID to the internal reference named catalog_master.

The subsequence occurrences will refer to that generated value and the ID resolver will simply substitute the generated ID for the references.

**External symbolic ID references**

XML elements that contain foreign key references may specify these using external symbolic references, in addition to the methods already mentioned.

External symbolic references are specified using the following format:

```
columnreference="@column1value@column2value@column3value..."
```

Where column1value, column2value and column3value are either literal values or XML entities, specifying the values for each of the column in the unique index for the table being referred to.

For example, assume that a contract with the attributes shown in Table 7-3 has been published. This contract can then be referenced from a child table, such as STORECNT or STOREDEF, using the following attribute value in the input to the ID resolver:

```
contract_id="@MyStoreContract@-2001@2@3@1"
```

<table>
<thead>
<tr>
<th>Column name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>MyStoreContract</td>
</tr>
<tr>
<td>MEMBER_ID</td>
<td>-2001</td>
</tr>
<tr>
<td>MAJORVERSION</td>
<td>2</td>
</tr>
<tr>
<td>MINORVERSION</td>
<td>3</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>1 (imported)</td>
</tr>
</tbody>
</table>

**Text transformer**

The text transformer is a utility for converting comma-delimited files to XML format. The utility is meant to be used to convert files to the format used by massloader and ID resolver.

The text transformer also has a graphical user interface, although this feature is only available in the WebSphere Commerce runtime. To start the graphical user interface, do the following:
1. Select **Start → Run...**

2. Enter `WC_installdir\bin\TextTrans.cmd` and click **OK**. For example, we entered:

   
   C:\WebSphere\CommerceServer56\bin\TextTrans.cmd

**Note:** We have experienced problems using the online help from within the graphical tool. Refer to the WebSphere Commerce Information Center for help on the tool. The help can be found under the following topic:

   Transforming data using the Text Transformation GUI tool

The WebSphere Commerce Information Center can be accessed from the following address:

   http://publib.boulder.ibm.com/infocenter/wc56help/index.jsp

If you wish to use the graphical user interface on a WebSphere Commerce Developer installation, you need to do the following:

1. Copy the following directory from WebSphere Commerce to WebSphere Commerce Developer:

   `WC_installdir\wcmadmin`

   To:

   `WCToolkit_installdir\wcmadmin`

2. Copy the following directory from WebSphere Commerce to WebSphere Commerce Developer:

   `WC_installdir\web`

   To:

   `WCToolkit_installdir\web`

3. Copy the following file from WebSphere Commerce to WebSphere Commerce Developer:

   `WC_installdir\bin\TextTrans.cmd`

   To:

   `WCToolkit_installdir\bin\TextTrans.cmd`

4. Edit the file `WCToolkit_installdir\bin\TextTrans.cmd` as shown in Example 7-4. The changed parts are highlighted in bold.

**Example 7-4**   TextTrans.cmd modified for WebSphere Commerce Developer

```
@echo off

REM     (C) Copyright IBM Corp. 2001, 2002
```
XML transformer

The XML transformer is a utility for transforming an XML document from one schema to another. The transformation is be specified using an XSL stylesheet. WebSphere Commerce uses the XML transformer extensively in its internal utilities, such as for the contract import tools and the access control policy loader.

Refer to the XSL family web site for more information about using XSL:

http://www.w3.org/Style/XSL/

Extractor

The **massextract** tool is a cross-platform utility for exporting data from a database to XML format. The output from the **massextract** tool is an XML file in the same format as the one used by the massloader utility, such as the one shown in Example 7-1 on page 119.

```bash
call setenv.bat
set lib=%WCS_HOME%\applications\wcfull.ear\lib\loader
set adminhome=%WCS_HOME%\wcmadmin
set log=%WCLOGDIR%\texttransfo.me.log

set clspath="%lib\wcmxmlp.jar;%adminhome\TextTransformerUI.zip;%adminhome\swing.jar"

:texttransfo.me
%JAVA_HOME%\bin\javaw -classpath %clspath% TextTransformerUI > %log%
```
Tip: As is the case with the remaining database-related tools, the `massextract` tool will use the current database configuration. For WebSphere Commerce Developer this means that the tools will use the current database settings as specified by the `setdbtype` command.

Example 7-5 on page 128 shows an example of an extraction filter to generate an XML document with the languages that are encoded using ISO88591-1 and Shift-JIS.

As the filter shown in Example 7-5 shows, the root element of a filter XML document must be `sqlx`. The root element contains a number of elements named `functionDef` and `execute`. Each `execute` element must correspond to a `functionDef` element, referenced through the `id` attributes. In Example 7-5, the `execute` element refers to the `functionDef` element by having the same value for the `id` attributes, namely `Language`.

Example 7-5 also shows how to use variables: The variable `:encoding` is specified using the `paramDef` subelement of the `functionDef` element and passed in using the `param` subelement of the `execute` element. As Example 7-5 shows, the variable is simply injected into the SQL specification, making it necessary to use quotes if the variable is a string. This is in contrast to the use of variable substitution in embedded SQL, where such quotes are not necessary.

Limitation: From the syntax, it would seem that it is possible to specify more than one `execute` element for each `functionDef` element. Using the scenario from Example 7-5, if it is required to extract the languages using Chinese encodings, it would be desirable to be able to specify the following two `execute` elements:

```xml
<execute id="Language">
  <param name=":encoding" value="Big5" />
</execute>
<execute id="Language">
  <param name=":encoding" value="gb2312" />
</execute>
```

This is unfortunately not possible. To do this, one would either have to modify the SQL in the `functionDef` element, or have two `functionDef` elements, one for each desired output encoding.
Example 7-5  Sample extraction filter to extract ISO8859-1 encoded languages

<?xml version="1.0"?>

<sqlx>
  <functionDef id="Language" schemaentity="language"
    description="Extract languages for a given encoding">
    <paramDef name=":encoding" description="Encoding" />
    <body>
      select * from "language" where encoding=':encoding'
    </body>
  </functionDef>

  <execute id="Language" description="Extract latin-based languages">
    <param name=":encoding" value="ISO8859-1" />
  </execute>
</sqlx>

Note: The massextract tool will place log files in the same directory as the binary file. For WebSphere Commerce Developer, this is:

   WCToolkit_installdir\bin

For WebSphere Commerce, the log files will be placed in:

   WC_installdir\bin

The log files are named:

- messages.txt
- trace.txt
- db2j.log (for IBM Cloudscape)

JSP batch compiler

The JSP batch compiler is a tool for ensuring that all JSP files are compiled before they are executed. Using the batch compiler is highly recommended, as it improves the user experience by avoiding long delays while a JSP file is compiled the first time it is used, as well as it allows for a faster detection of any static errors in the deployed JSPs.

The JSP batch compiler can be used both in a runtime environment and in the development environment. In the development environment, the batch compiler is invoked as follows:

   WCToolkit_installdir\bin\JspBatchCompiler <test_environment_type>
Where `<test_environment_type>` is the type of test environment to use to compile the JSP for. It can be one of:

- `light`
- `full`

The test environment must be specified, as the JSP batch compiler invokes the same JSP compiler that would have been invoked if the JSP had been activated through that test environment. Also, the two test environments do not share the temporary directories in which the compiled code would be placed.

In a runtime environment, the JSP batch compiler is invoked as follows:

```
WAS_inslalldir\bin\JspBatchCompiler -enterpriseapp.name WC_<instance_name> -webmodule.name Stores.war -cell.name <hostname> -node.name <hostname> -server.name WC_<instance_name>
```

Where:

- `<instance_name>` is the name of the WebSphere Commerce instance
- `<hostname>` is the host name of the runtime node

For example, we used the following command:

```
C:\WebSphere\AppServer\bin\JspBatchCompiler -enterpriseapp.name WC_demo -webmodule.name Stores.war -cell.name wcdemo -node.name wcdemo -server.name WC_demo
```

**Tip:** The server does not need to be running when the JSP batch compiler is invoked, no matter if the batch compiler is invoked on a WebSphere Commerce Developer or a WebSphere Commerce runtime.

Also, since the JSP batch compiler uses the web application’s configuration, customized code will be picked up by the compiler just as when the JSP is invoked through the application server, or test environment, from a browser.

**Encryption tools**

In order to generate encrypted values, WebSphere Commerce contains tools for encrypting values using the various methods in use by the WebSphere Commerce framework.

- `wcs_encrypt`
- `wcs_password`
- `wcs_pmpassword`
Please note that WebSphere Commerce only contains script for encryption. There are currently no standard WebSphere Commerce command line utilities for decryption of encrypted values.

**wcs_encrypt**

This tool is a front-end to the WebSphere Commerce class com.ibm.commerce.util.nc_crypt (see 8.6.2, “Encrypt sensitive information” on page 180) and can thus be used to either encrypt a string using a 64-bit key, or to obfuscate a string.

**Important:** Obfuscation is typically used to avoid that sensitive information, such as the merchant key, database passwords, etc. stored in WebSphere Commerce configuration files are instantly readable. It is important to stress that the information should in no way be considered safe just because it is not looks that way. Obfuscation is used to avoid that a casual onlooker can pick up such information from a glance on a terminal where the information is presented.

Care should be taken to protect files, containing such sensitive information, using file system protection and other available operating system mechanisms. See also 8.6.3, “Protect sensitive configuration files” on page 181.

Since the **wcs_encrypt** command has two functions, it can be invoked in two different ways:

1. **Obfuscate a value, such as the merchant key:**
   
   ** wcs_encrypt ** <text>
   
   **Example:**
   
   wcs_encrypt deadbeef12345678
   
   **Output:**
   
   ASCII encrypted string: wJNBcU/5M0DwO6RLTuRtcvC7xiUN2sAc
   HEX encrypted string: 774A4E4263552F354D3044774F62524C54755274637643377869554E32734163
   
   **Note:** Even though the output says ASCII encrypted string, the output is really not encrypted. It is possible to retrieve the original text, in this case deadbeef12345678, without an encryption key, since the algorithm is known.

2. **Encrypt a value using a merchant key:**
   
   wcs_encrypt <text> <64-bit hexadecimal merchant key>
Example:
  wcs_encrypt password deadbeef12345678

Output:
  ASCII encrypted string : /WzGMpIK7Z5H9jpXTQzPPA==
  HEX encrypted string : 2F577A474D70494B375A3548396A705854517A5050413D3D

The output from the wcs_encrypt command, as shown after the text ASCII encrypted string, can then be stored in a configuration file, as the case typically is for obfuscated values, or in the database, as the case is for encrypted values.

The ASCII encrypted string is in fact a base 64 encoding of the encrypted, or obfuscated, value, as represented by the HEX encrypted string.

wcs_password
In earlier versions of WebSphere Commerce, user passwords would simply be encrypted using the merchant key and stored in the database. Since WebSphere Commerce V5.4, however, the algorithm has also involved a one-way hashing with a random salt value. In order to be able to generate encrypted passwords, the wcs_password command was introduced.

The syntax of the wcs_password command is:
  wcs_password <password> <salt> <64-bit hexadecimal merchant key>

Where:
  <password> is the password to be encrypted.
  <salt> is a random ASCII text string.
  <64-bit hexadecimal merchant key> is the merchant key to use to encrypt the password with.

For example, we used the following command:
  wcs_password password 8nmbfk5z5jw8 deadbeef12345678

Resulting in the following output:
  ASCII Format: eQm5AiZNqS5vpxIMs/4qfE1eruZtrvD8RLUSuzS1k=
The random salt and the ASCII formatted string can then be inserted to the USERREG table to reset the password to the specified value.

**Limitation:** The `wcs_password` command is only included in WebSphere Commerce runtime and are thus not available in WebSphere Commerce Developer. If you need to encrypt password in the development environment, you can use the script shown in Example 7-6. Place the script in the directory `WCToolkit_installdir\bin`.

### Example 7-6  wcs_password.bat script for WebSphere Commerce Developer

```batch
@echo off
%~d0
cd %~p0
setlocal

call setenv
set CP=%RUNTIME_CLASSES%;%PROPERTIES%;%RUNTIME_JARS%\wcs.jar;

%JAVA_HOME%\bin\java -classpath %CP% com.ibm.commerce.util.WCSPassword %1 %2 %3
endlocal
```

**wcs_pmpassword**

The `wcs_pmpassword` command is used to generate passwords for WebSphere Commerce Payments. The syntax of the command is:

```
wcs_pmpassword <password> <salt>
```

Where:

- `<password>` is the password to be encrypted.
- `<salt>` is a random ASCII text string. Since the `wcs_pmpassword` command is typically used to generate a payments password for a given WebSphere Commerce user, this is typically just taken from the SALT column of the USERREG table.

For example, we used the following command:

```
wcs_pmpassword password 8nmbfk5z5jw8
```
Resulting in the following output:

ASCII Format: bJTFxYhozbOHfPaLOvoceXH7cgU=
Hex Format: 624A54467859686F7A624F486650614C305666365584837637553D

**Limitation:** The `wcs_pmpassword` command is only included in WebSphere Commerce runtime and are thus not available in WebSphere Commerce Developer. If you need to encrypt password in the development environment, you can use the script shown in Example 7-7. Place the script in the directory `WCToolkit_installdir\bin`.

**Example 7-7 wcs_pmpassword.bat script for WebSphere Commerce Developer**

```batch
@echo off
%~d0
cd %~p0
setlocal

call setenv
set CP=%RUNTIME_CLASSES%;%PROPERTIES%;%RUNTIME_JARS%;wcs.jar;

%JAVA_HOME%in\java -classpath %CP% com.ibm.commerce.util.WCSPMPassword %1 %2

dendlocal
```

**Database cleanup utility**

The database cleanup utility, `dbclean`, should be used on a regular basis to clean up the production database and file system. The utility is configured by default to for a number of cleanup type operations using predefined rules. For example, by the `dbclean` command contains the following predefined rules for cleaning up orders shown in Table 7-4.

**Note:** Although named `dbclean`, the command is also capable of deleting file system resources. This is done by specifying an SQL `select` statement in the configuration for `dbclean`. The utility will then locate the corresponding file assets and delete them.

As is indicated in the rules listed in Table 7-4, it is possible to specify parameters when invoking `dbclean`. The `-days` parameter specifies how old the objects should be before they are deleted, while `-name` parameter can be used to specify the name, or identifier, of the objects to delete. How the `-days` and `-name` parameters are used, depends on the actual cleanup rule.
For example, none of the rules shown in Table 7-4 use the `-name` parameter, while all use the `-days` parameter to ensure that orders have a certain age before they are purged.

**Table 7-4  List of pre-defined rules for cleaning orders**

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canceled</td>
<td>Delete all canceled orders (orders in state X), older than the specified number of days, for which a return or store credit is not registered. Also delete any business flow control information for non-existent orders.</td>
</tr>
<tr>
<td>completed</td>
<td>Delete all completed orders (orders in state C), older than the specified number of days, for which a return or store credit is not registered. Also delete any business flow control information for non-existent orders.</td>
</tr>
<tr>
<td>deposited</td>
<td>Delete all deposited orders (orders in state D), older than the specified number of days, for which a return or store credit is not registered. Also delete any business flow control information for non-existent orders.</td>
</tr>
<tr>
<td>quotation</td>
<td>Delete all orders that represent the child in a parent-child quotation relation, where the relation has been marked for deletion. Also, delete the relation.</td>
</tr>
<tr>
<td>shipped</td>
<td>Delete all shipped orders (orders in state S), older than the specified number of days, for which a return or store credit is not registered. Also delete any business flow control information for non-existent orders.</td>
</tr>
<tr>
<td>stale_guest</td>
<td>Delete all guest user orders that are either pending, submitted, waiting for approval or with approval denied (orders in state P, I, W or N), older than the specified number of days, for which a return or store credit is not registered and for which no inventory movement has been initiated. Also delete any business flow control information for non-existent orders.</td>
</tr>
<tr>
<td>stale_non_guest</td>
<td>Delete all orders for registered users that are either pending, submitted, waiting for approval or with approval denied (orders in state P, I, W or N), older than the specified number of days, for which a return or store credit is not registered and for which no inventory movement has been initiated. Also delete any business flow control information for non-existent orders.</td>
</tr>
</tbody>
</table>
7.2 Migration tools

WebSphere Commerce V5.6 contains a number of tools for assisting a migration from a previous version of WebSphere Commerce. The individual tools are described in the following sections.

7.2.1 Database migration (migratedb)

The migratedb command migrates the standard WebSphere Commerce database schema to the level used in WebSphere Commerce V5.6. If the existing site contains customized tables, these must be manually migrated.

**Important:** Any customizations of the database schema must be handled manually. If, for example, a customized table has been added with a referential constraint to a standard WebSphere Commerce table, that referential constraint must be dropped from the database before the migration starts.

The database migration script will perform an in-place migration of the database, that is to say the script will transform a given database from a previous WebSphere Commerce database schema version to the WebSphere Commerce V5.6 database schema.

The migratedb script migrates the database schema and the unencrypted data as needed. The behavior of this script is controlled by one or other configuration files depending on your pre-migration WebSphere Commerce level.

For example, if you are currently using WebSphere Commerce Suite V5.1, the configuration file used is WC56_install_dir/schema/migration/config/DataMigrationPlan51.xml. Example 7-8 shows an excerpt of that configuration file.

**Example 7-8  Excerpt of DataMigrationPlan51.xml file**

```xml
<dataMigrationCommand name="drop.wcs.constraint" type="sql">
  <command>
    wcs.drop.referential.sql,
    wcs.drop.view.sql,
    wcs.drop.key.sql,
    wcs.drop.index.sql
  </command>
</dataMigrationCommand>
<dataMigrationCommand name="backup.wcs.bootstrap" type="sql">
  <command>
    wcs.backup.bootstrap.sql,
    wcs.delete.bootstrap.sql,
  </command>
</dataMigrationCommand>
```
<dataMigrationCommand name="apply.wcs.schema.delta" type="sql">
  <command>
    wcs.fix.null.notnull.sql,
    wcs.schema.new.tables.sql,
    wcs.schema.recreate.tables.sql,
    wcs.schema.alter.tables.sql
  </command>
  <property name="commitCount" value="1" />
</dataMigrationCommand>

This file controls the sequence of the execution of various SQL files, in addition to other activities. This provides fine-grained control over how migration should proceed. In the following are some examples of the usefulness of this approach.

Some of the column lengths have decreased in the WebSphere Commerce V5.6 schema when compared to the column lengths in database schemas for previous WebSphere Commerce versions. However, if you have data that occupies more space than is allowed by column length in WebSphere Commerce V5.6, then the data migration schema will fail.

We strongly recommend that you re-evaluate your need to have larger column length and solve the issue in a different way, for example by using a different column to hold this data. However, if you must have the same column length as defined in the previous schema version, then you can change the behavior of one of the SQL files mentioned in the configuration file.

Both the standard output and the migration log file specify the command that it is currently executing. This can be used for error resolution. For example, if an error occurs while creating the new WebSphere Commerce V5.6 tables, the log file will show that the last executed command before the error message will be wcs.schema.recreate.tables.sql. For WebSphere Commerce Suite V5.1, this file can be found in the directory WC56_install_dir/schema/migration/51.

Even if no error occurs, you can utilize the logging of the current SQL command to optimize the execution times. If, for example, a given step is taking a long time to run, you can examine the corresponding SQL statements and tune your database to run this SQL command more efficiently.
7.2.2 Encrypted data migration (migrateEncryptedInfo)

The `migrateEncryptedInfo` script transforms encrypted data in a database that has already had its schema migrated by the `migratdb` script. The encrypted data must be migrated to handle new algorithms for encrypting passwords and to support changing of the merchant key, if needed.

The `migrateEncryptedInfo` script reads all the rows of a given table. Then it processes and commits a subset of rows at a time. Note that the instance must be migrated prior to running this script since, in addition to other instance specific files, the script needs the migrated `instance_name.xml` file.

For migrations from WebSphere Commerce Suite V5.1 to WebSphere Commerce V5.6, this script will invoke the `MKChangeApp` command twice with different parameters and configuration files:

1. The first invocation uses the `CCInfoDBUpdate.dbtype.txt` configuration file, which defines the following two action items:
   - System key to current key migration: Prior to WebSphere Commerce Suite V5.1.1.2, the merchant key was not used to encrypt credit card data. Instead a system key was used. Since this behavior has changed, the data will have to be re-encrypted.
   - PDI flag enforcement: The variable `PDIEncrypt` in the instance file indicates whether credit card data is encrypted. Depending on the value of this variable, credit card information is encrypted or decrypted accordingly. This script provides you with an opportunity to change and enforce the PDI encryption setting.

2. The second invocation of the `MKChangeApp` command uses the configuration file `schema\dbtype\migration\DBUpdate.txt`, which defines the following two actions:
   - Password migration: Passwords are re-encrypted using the random salt values and one-way hash keys.
   - Merchant key migration: This script gives you the opportunity to change your merchant key and migrate your data based on the new merchant key.

This script acts on the tables listed in Table 7-5 on page 138.
### Table 7-5  Tables used by migrateEncryptedInfo script file

<table>
<thead>
<tr>
<th>TABLE</th>
<th>COLUMNS UPDATED</th>
<th>UNIQUE ID</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERREG</td>
<td>LOGONPASSWORD, SALT</td>
<td>USERS_ID</td>
<td>Password related information</td>
</tr>
<tr>
<td>ORDERPAYINFO</td>
<td>VALUE</td>
<td>ORDERPAYINFO_ID</td>
<td>Credit Card related information</td>
</tr>
<tr>
<td>ORDPMTHD</td>
<td>PAYDEVICE</td>
<td>ORDERS_ID, PAYMETHOD, PAYDEVICE, REFUNDNUMBER</td>
<td>Credit card related information</td>
</tr>
<tr>
<td>PATTRVALUE</td>
<td>STRINGVALUE ENCRYPTFLAG</td>
<td>PATTRVALUE_ID</td>
<td>Personalization attributes, handled with credit card data processing (table began to exist in v5.4 onwards)</td>
</tr>
</tbody>
</table>

### 7.2.3 Instance migration (wcim)

The instance migration script supports moving of specific file assets and to migrate the instance configuration.

Here is a brief description about what **wcim** does at the high level. The following two steps are additional for switch-over instance migration (for a runtime environment only). You do not need to do these before you migrate a local instance:

1. Package **wcim** on a V5.6 machine and its environment for remote execution. Run **wcim** to create a ZIP file, including **wcim** itself, an the environment it requires. This package is used to build the migration environment for the switch-over migration.

2. Unpack **wcim** on V5.1 machine. Unzip the **wcim** ZIP package generated earlier. This builds the runtime environment with the correct level of the Java Developer Kit (JDK), Extensible Markup Language (XML) parser, ANT, Configure Manager client, and **wcim** while maintaining the file permissions and ownership.

**Note:** Since the package that is created includes all these platform dependent files, scripts, and attributes, you cannot migrate across platforms.

For example, to migrate a local WebSphere Commerce Suite V5.1 instance or a WebSphere Commerce Studio V5.1 instance, follow these steps:
1. Back up the instance to be migrated. Backing up the WebSphere Commerce runtime environment is required in case of switch-over instance migration. While this is optional, we recommend that you do it in other instances. The backup includes:

- Configuration and installation information about the previous WebSphere Commerce installation
- Configuration information about the WebSphere Commerce instance to be migrated
- Configuration information about the local IBM HTTP Server.
- Configuration information about WebSphere Commerce Payments, the supported payment cassettes, and the deployed EAR file

If there is no EAR file, `wcim` collects enough information to build a migrated EAR file. All files are compressed into a single ZIP backup-instance.zip file to be transferred to the target WebSphere Commerce server.

2. Migrate the instance. This includes:

- Creating a migrated EAR file based on the original EAR file, and the default EAR in the new release
- Wrapping stores published before the migration into a Web module, with the addition of multiple Web modules and resource adapters
- Enterprise JavaBeans (EJBs) for the correct database type, JavaServer Pages (JSPs), servlets, commands, HTML files, images from the new version packaged into the migrated EAR file
- An application server named `WC_instance_name` is created with the correct configuration required to run the migrated EAR file
- The JDBC provider and the data source configured for the server, with the migrated EAR file installed onto the server
- Modifying the deployment descriptor
- Migrating the configuration of the original instance, including the instance itself, access control, organization, trading, loader, payment, and store JSPs
- Changing the Web server host name and the database name/SID by `wcim` during the migration if users choose to do so
- For the switch-over migration, configuring the correct Web server host name, WebSphere Application Server node name, and WebSphere Application Server cell name for the migrated instance
- For local replace migration, assuming the same Web server host name.

The user can take this opportunity to switch to a different Web server. For local coexistence migration, a different Web server host name is enforced.
Local IHS is configured. WebSphere Application Server plug-in configuration is regenerated. You can run the backup and migration steps together for local migration. No human intervention is required in between.

- For local coexistence migration, or migration from WebSphere Commerce V5.5 replace migration, using a temporary host name (required for migration)

wcim can perform IP-switching to reconfigure the migrated instance to use the final Web server host name. WebSphere Application Server virtual hosts, instance configuration XML file, and local IHS configuration are modified. WebSphere Application Server plug-in configuration is regenerated.

3. The development environment is migrated by wcim in a similar fashion as explained in steps 3 and 4. In addition, custom store assets, servlets, and commands are moved to the following pre-configured projects in the WebSphere Studio Application Developer workspace:

- Stores
- WebSphereCommerceServerExtensionLogic

Note: The wcim tool converts all occurrences of `<%@ page language="JAVA" %>` to `<%@ page language="java" %>` during this step. However, during migration projects, we have found that not all occurrences are modified. After this step, ensure that no more occurrences of `<%@ page language="JAVA" %>` exists. You can do this by doing a file content search in the Stores project.

### 7.3 Additional tools

The development and migration tools that have already been discussed in this chapter are all tools that are shipped as part of the WebSphere Commerce V5.6 product.

In addition to these tools, there are a number of unsupported tools that have previously been used with success in customer engagements:

- Lightweight Object Query System
- Auto-deploy tool
- CommerceQA test tool

Although these tools are not officially supported by IBM, they can provide added value and speed up the development process, and as such they deserve mention.
### 7.3.1 Lightweight Object Query System

While traditionally a part of the Transition Tools Suite (TTS, a migration tool suite for migrating from net.data based versions of WebSphere Commerce), the lightweight object query system (LOQS) is a useful tool, not only for migration efforts from net.data, but also for developing new sites.

**Note:** LOQS is currently not available for download. Contact your IBM representative for more information about how to obtain LOQS.

LOQS consists of a runtime component and a set of development tools and provides the ability to automate the process of generating session beans for read-only database access, as well as the generation of lightweight data objects to hold the retrieved data.

The LOQS suite contains the following two tools:

- **LOQS wizard**
- **LOQS command line generator**

Each of these will be briefly described in the following sections.

**LOQS wizard**

The LOQS wizard is a graphical user interface (GUI) interface for building session beans for SQL specified by the user, as well as the related data objects.

The wizard can be used to either generate XML configuration files, called LOQSML files, for later use by the LOQS command line generator, or directly generate the Java code.

**LOQS command line generator**

The LOQS command line generator, `loqsgen`, is used to generate Java code from a previously prepared LOQSML file. The LOQSML file can either be generated by the LOQS wizard, or written by hand, or both.

The LOQS command line generator is invoked as follows:

```
loqsgen <LOQSML file> [-v]
```

Where `<LOQSML file>` is the name of the LOQSML file. The generated Java code will be placed in the directory as specified in the LOQSML file.
7.3.2 Auto-deploy tool

In order to automate many of the repetitive and manual steps necessary when deploying assets to a WebSphere Commerce V5.6 runtime, IBM has developed a tool that can build a package containing assets and executable code such that the package itself can be used to deploy a variety of assets, such as:

- Database schema changes.
- Database content changes.
- JSP files.
- Property files.
- Java code for data beans, task and controller commands.
- EJB code.

The auto-deploy tool is described in detail in the following developerWorks article from where the tool can also be downloaded:


7.3.3 CommerceQA test tool

It is important to establish as systematic unit testing environment. In order to support this, IBM has established a set of sample files that illustrate how to use HttpUnit to test WebSphere Commerce sites.

HttpUnit is an open source Java utility that can be used to emulate a browser. Since it can be invoked from Java, it can be used in combination with a testing framework, such as JUnit, to automate web page testing. HttpUnit supports cookies and HTTP authentication, allowing it to be used to test all pages of a WebSphere Commerce site.

The CommerceQA tool is described in detail in the following developerWorks article from where the tool can also be downloaded:


HttpUnit is available from the following address:

http://httpunit.sourceforge.net/

JUnit is available from the following address:

http://junit.sourceforge.net/
Chapter 8. Developing new sites

This chapter describes best practices for implementing new WebSphere Commerce sites. The focus of this chapter is on the techniques and best practices considered during the build phase, thus the chapter will be of a more hands-on nature than the rest of the book.

The chapter is organized into the following sections:

- Basing development on a starter store
- Use separate database server
- EJB and data bean best practices
- Best practices for JSP programming
- Command best practices
- General security considerations
8.1 Basing development on a starter store

WebSphere Commerce contains several fully functional starter stores. We recommend that you use one of these stores as a basis for new development.

The starter stores may be modified as necessary for your business needs. Refer to the WebSphere Commerce Information Center for detailed information about the starter stores and for guidelines on modifying specific parts of the stores.

Starter stores are meant to shorten the time of installing WebSphere Commerce to going live with your site. This is achieved by not only providing commands and JSP files but also the appropriate organization layout, approval configuration, and subscription to the appropriate access control policy groups for particular business scenarios. Each starter store is packaged as a WebSphere Commerce store archive. Store archives contain the assets to create a functional store, no further installation is necessary. In order to create a running store, you must publish one of the composite store archives. The lightweight test environment in WebSphere Commerce Developer V5.6 (see 7.1.1, “Lightweight WebSphere Commerce test environment” on page 116) is preconfigured with published starter stores.

The WebSphere Commerce Business Edition contains starter stores for the following business models:

- **Consumer direct (ConsumerDirect)**
  
  A business-to-consumer (B2C) business model for supplying sales, services and information channels directly between a retailer and its customers. Note that in this context, a retailer is a role defined as a business that sells merchandise directly to consumers, and does not necessarily a traditional retailer, but could be a manufacturer, or even a government agency.

- **B2B direct (B2BDirect and AdvancedB2BDirect)**
  
  The business-to-business (B2B) direct model is used for handling the sales, services and information between two businesses.

- **Demand chain (DemandChain, aka. Commerce Plaza)**
  
  The demand chain model provides a way for a manufacturer to connect its resellers to its distributors. Each reseller can set up a hosted store in the demand chain hub, using a subset of the manufacturer's product catalog.

- **Supply chain (SupplyChain, aka. Supplier Hub)**
  
  The supply chain model provides a way for a manufacturer to connect its resellers to its distributors. Each reseller can set up a hosted store in the demand chain hub, using a subset of the manufacturer's product catalog.

- **Hosting (CommerceHostingHub)**
The hosting model allows for a business, typically an ISP, to provide store hosting services, as well as the ability to generate a store directory, much like a mall.

**Limitation:** WebSphere Commerce Professional Edition and WebSphere Commerce Express only support the consumer direct and B2B direct store models, via the following two starter stores:

- ConsumerDirect
- B2BDirect

It is very important to understand the features and limitations of the many starter stores, in order to be able to make an informed decision about which starter store to base the development of a new site upon. Refer to the WebSphere Commerce Information Center to enhance your understanding about the business processes for the various starter stores.

### 8.2 Use separate database server

Deploying a separate database server to create a two-tier or three-tier topology, as illustrated in Example 8-1 and Example 8-2 on page 146 respectively, is one of the most effective methods of implementing a scalable architecture.

*Figure 8-1 A two-tier architecture with separate database*
Figure 8-2  A two-tier architecture with separate database

In both Example 8-1 and Example 8-2, the Database Server is located in the Application zone and is protected by firewalls between the DMZ and the companies intranet. Some companies choose not to have their WebSphere Commerce sites hosted by a third party, instead hosting the solution in-house. In these situations, the intranet zone can typically access the application zone, while the DMZ is not directly accessible. In such situations it can be necessary to have a separate web server (named Administration Web Server in the figures) in order to access the administrative WebSphere Commerce applications.

In situations where the administrative users access the application from the internet, or when access to the DMZ from the intranet is available, such an extra web server is not necessary.

### 8.3 EJB and data bean best practices

This section describes best practices for developing new WebSphere Commerce EJBs and data beans. The section contains information about the following topics:

- Customizing WebSphere Commerce tables
- Customizing WebSphere Commerce EJBs
- SQL security considerations
- Use well-formed ejbCreate constructors
- Lazy fetch data retrieval
- Handling database vendor specific code
8.3.1 Customizing WebSphere Commerce tables

If it is determined that the existing WebSphere Commerce database schema is not sufficient, there are two ways to add data to a specific WebSphere Commerce entity:

► Use existing customizable columns (FIELD1, FIELD2, FIELD3, etc.)
► Add a new customized table, referencing the standard table.

**Important:** We strongly discourage any modifications to existing WebSphere Commerce tables for the reasons noted below.

If the existing customizable fields are sufficient for the customization requirements, an extra table should *not* be created. The three main reasons for this are:

► Using existing customizable fields require practically no additional development effort.
► Future migration efforts are greatly increased when custom tables are added to the database schema, since this extends the number of manual steps involved in the migration.
► The performance of the site is impacted when additional tables are introduced, as for each standard WebSphere Commerce EJB lookup, an additional lookup must be performed to retrieve the data from the custom table.

When creating new customized tables for the WebSphere Commerce schema, the new tables should be prefixed with an X to distinguish the tables from standard WebSphere Commerce tables. This will also minimize the risk of having a name clash when the site is migrated to a future version of WebSphere Commerce.

The new table should use the same primary key as the standard WebSphere Commerce table that it will refer to. This primary key should then also be declared as a foreign key to the existing WebSphere Commerce table.
8.3.2 Customizing WebSphere Commerce EJBs

If the creation of additional tables is required for the WebSphere Commerce database schema as discussed in 8.3.1, “Customizing WebSphere Commerce tables” on page 147, a new CMP EJB, mapping that new table, must be created.

New EJBs must be created in the WebSphereCommerceExtensionsData project in the WebSphere Commerce Developer workspace.

Important: Although it is possible to modify one of the existing WebSphere Commerce EJBs to also retrieve data from the customized table, this is strongly discouraged for maintenance reasons. Specifically the application of future fix packs and migrations will be severely complicated if this is done.

8.3.3 SQL security considerations

A particular coding practice for the use of EJBs and data beans minimizes the chance for malicious users to access your database in an unauthorized manner. SQL statements should be created at development time and not built dynamically. Use SQL parameters to gather run-time input information.

Suppose a session bean that retrieves data for a data bean. This session bean has a method, findOrdersByComment that retrieves all orders for a given user that have a given order comment. The method will be called from a command using the user ID of the current user. This should ensure that order for other users are not visible to the current user.

Example 8-1 shows one way to implement the SQL for the findOrdersByComment method. This approach builds the SQL from the parameters and then executes that SQL.

This approach, however, has a serious flaw in that it creates an SQL injection vulnerability whereby it exposes the site to unauthorized access.

Example 8-1   Building SQL dynamically using the input parameters

```java
public class mySessionBean
    extends com.ibm.commerce.base.helpers.BaseJDBCHelper
    implements SessionBean
{
    public Collection findOrderIdsByComment(String comment, Long ownerId)
        throws javax.naming.NamingException, SQLException
    {
        ... 
        String sql = "select orders_id from orders where comments='"+comment+"' and member_id="+ownerId;
```
PreparedStatement stmt = getPreparedStatement(sql);
ResultSet rs = executeQuery(stmt, false);
...
}
}

If, for example, the user enters the following search text:

' || comments--'

The resulting SQL will be:

select * from orders where comments='' || comments--' and member_id=<user_id>

Where <user_id> is the user ID of the current user.

This SQL is equivalent to, and can be reduced to, the following SQL:

select * from orders where comments is not null

And thus all orders, regardless of which user placed that order, are returned. All it requires to exploit this is a little knowledge about the database model and the query being executed. The database model is known and the nature of the query can in most cases be deduced by injecting other query strings and examining the returned error messages.

Example 8-2 shows an example of implementing the method using SQL parameters instead. This is the recommended way to retrieve information from the database, as it is not vulnerable to the same sort of attack as the code in Example 8-1.

Example 8-2   Using SQL parameters to inject variables to pre-built SQL

```java
public class mySessionBean
extends com.ibm.commerce.base.helpers.BaseJDBCHelper
implements SessionBean
{
    public Collection findOrderIdsByComment(String comment, Long ownerId)
        throws javax.naming.NamingException, SQLException
    {
        String sql = "select orders_id from orders " +
            "where comments=? and member_id=?";
        PreparedStatement stmt = getPreparedStatement(sql);
        stmt.setString(1, comment);
        stmt.setLong(2, ownerId);
        ResultSet rs = executeQuery(stmt, false);
        ...
    }
```
8.3.4 Use well-formed ejbCreate constructors

Ensure the ejbCreate method for each CMP EJB contains a parameter for each underlying column in the table that the bean maps to. Also ensure that those ejbCreate methods have been promoted to the remote interface.

Doing so will ensure that a row for that table can be created in one operation. Otherwise, a user of the bean will have to first create the row and then update the remaining columns. This will result in two database accesses, and thus unnecessary overhead.

8.3.5 Lazy fetch data retrieval

When a data bean is activated, it can be populated by a data bean command or by the data bean's populate method. The attributes that are retrieved come from the data bean's corresponding entity bean. An entity bean may also have associated objects, which themselves, have a number of attributes.

If, upon activation, the attributes from all the associated objects were automatically retrieved, a performance problem may be encountered. Performance may degrade as the number of associated objects increase.

Consider a product data bean that contains a large number of cross-sell, up-sell or accessory products (associated objects). It is possible to populate all associated objects as soon as the product data bean is activated. However, populating in this manner may require multiple database queries. If not all attributes are required by the page, multiple database queries may be inefficient.

In general, not all attributes are required for a page, therefore, a better design pattern is to perform a lazy fetch as illustrated in Example 8-3.

Example 8-3  Example of lazy fetching

```java
public ProductDataBean[] getCrossSellProducts()
{
    if (crossSellDataBeans == null)
        crossSellDataBeans = getCrossSellDataBeans();
    return crossSellDataBeans;
}
```
### 8.3.6 Handling database vendor specific code

Although SQL is standardized, it is often necessary to resort to database vendor specific code. An example is the method to retrieve the current timestamp:

- IBM DB2 Universal Database: `CURRENT_TIMESTAMP` or `CURRENT_TIMESTAMP`.
- IBM Cloudscape: `CURRENT_TIMESTAMP` or `CURRENT_TIMESTAMP`.
- Oracle 9i: `SYSDATE`.

To help in such situations, the class used as a superclass for all custom session EJBs, `com.ibm.commerce.base.helpers.BaseJDBCHelper`, contains several static utility methods to check for the current database vendor, as shown in Table 8-1.

**Table 8-1 Database vendor methods in BaseJDBCHelper**

<table>
<thead>
<tr>
<th>Method</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>useDB2</td>
<td>Returns whether IBM DB2 Universal Database is used (regardless of hardware platform).</td>
</tr>
<tr>
<td>useDB2_390</td>
<td>Returns whether IBM DB2 Universal Database is used on IBM zSeries.</td>
</tr>
<tr>
<td>useCloudscape</td>
<td>Returns whether Oracle 9i is used.</td>
</tr>
<tr>
<td>useOracle</td>
<td>Returns whether IBM Cloudscape is used.</td>
</tr>
</tbody>
</table>

In addition, the class `com.ibm.commerce.utils.TimestampHelper` contains methods to retrieve the SQL string used to retrieve the current timestamp, as shown in Table 8-2.

**Table 8-2 Retrieving SQL for timestamp manipulation from the TimestampHelper class**

<table>
<thead>
<tr>
<th>Method</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getSQLCurrentTimestamp</code></td>
<td>Returns a string with the database vendor specific SQL for retrieving the current timestamp.</td>
</tr>
<tr>
<td><code>getSQLSyntaxTimestampPlusInteger</code></td>
<td>Given a string representation of a timestamp and an integer value, this method returns the SQL string for adding the integer value, interpreted as a number of seconds, to the specified timestamp.</td>
</tr>
</tbody>
</table>

Example 8-4 shows an example of using the `TimestampHelper` class to generate database vendor neutral code.
Example 8-4  Handling database vendor specific code

```java
public class mySessionBean
    extends com.ibm.commerce.base.helpers.BaseJDBCHelper
    implements SessionBean
{
    private final static Integer SECONDS_30_DAYS = new Integer(-60*60*24*30);
	public Collection findOrdersLast30Days(Long ownerId)
	    throws javax.naming.NamingException, SQLException
	{
		...
		final String sqlNow = TimestampHelper.getSQLCurrentTimestamp();
		final String sql30daysago =
		    TimestampHelper.getSQLSyntaxTimestampPlusInteger(
		        sqlNow, SECONDS_30_DAYS);
		String sql = “select orders_id from orders “ +
		    “where timeplaced <= “+sql30daysago;
		PreparedStatement stmt = getPreparedStatement(sql);
		ResultSet rs = executeQuery(stmt, false);
		...
	}
}
```

8.3.7 Using session beans

One of the strengths of WebSphere Commerce stems from its ability to take advantage of container-managed persistence (CMP) entity beans. CMP entity beans are distributed, persistent, transactional, server-side Java components that can be generated by the tools provided by WebSphere Studio Application Developer. In many cases, CMP entity beans are an extremely good choice for object persistence and they can be made to work at least as efficiently or even more efficiently than other object-to-relational mapping options. For these reasons, WebSphere Commerce has implemented core commerce objects using CMP entity beans.

There are, however, some situations in which it is recommended to use a session bean JDBC helper. These situations include the following:

- A case where a query returns a large result set. This is referred to as the large result set case.
- A case where a query retrieves data from several tables. This is referred to as the aggregate entity case.
- A case where an SQL statement performs a database intensive operation, or queries that return data that does not correspond to existing CMP EJ Bs. This is referred to as the arbitrary SQL case.
Large result set case

There are cases where a query returns a large result set and the data retrieved are mainly for read or display purpose. In this case, it is better to use a stateless session bean and within that session bean, create a finder method that performs the same functions as a finder method in an entity bean. That is, the finder method in the stateless session bean should do the following:

1. Perform an SQL select statement.
2. For each row that is fetched, instantiate an access bean.
3. For each column retrieved, set the corresponding attributes in the access bean.

When the access bean is returned, the command is unaware of whether the access bean was returned by a finder method in a session bean or from a finder method in an entity bean. As a result, using a finder method in a session bean does not cause any change to the programming model. Only the calling command is aware of whether it is invoking a finder method in a session bean or in an entity bean. It is transparent to all other parts of the programming model.

Aggregate entity case

In this case, one view combines parts of several objects and a single display page is populated with pieces of information that come from several database tables. For example, consider the concept of *My Account*. This may consist of information from a table of customer information (for example, the customer name, age and customer ID) and information from an address table (for example, an address made up of a street and city).

It is possible to construct a simple SQL statement to retrieve all of the information from the various tables by performing an SQL join. This can result in significant performance gains, and is referred to as performing a *deep fetch*. Example 8-5 shows an SQL select statement for a deep fetch of data from the `CUSTOMER` and `ADDRESS` tables.

**Note:** If the session bean is being used as a JDBC wrapper to retrieve information from the database, it becomes more difficult to implement resource-level access control. When a session bean is used in this manner, the developer of the session bean must add the appropriate `where` clauses to the `select` statement in order to prevent unauthorized users from accessing resources.

The trade-off between performance and maintainability should therefore be carefully considered.
The tools in WebSphere Studio Application Developer for enterprise beans at the EJB 1.1 specification do not support this notion of a deep fetch. Instead, a lazy fetch, resulting in an SQL select for each associated object, is used. This is not the preferred method for retrieving this type of information.

In order to perform a deep fetch, it is recommended that you use a session bean. In that session bean, create a finder method to retrieve the required information. The finder method should do the following:

1. Perform an SQL select statement for the deep fetch.
2. Instantiate an access bean for each row in the main table as well as for each associated object.
3. For each column fetched and for each associated object fetched, set the corresponding attribute in the access bean.

In some cases where an access bean contains a getter method that returns another access bean, the generated code for the getter method does not correctly cache the return value. To amend this, you can extend the access bean class and create a local copy of the returned access bean, as shown in Example 8-6.

**Example 8-6 Overriding potential non-cached accessor methods**

```java
public class CustomerAccessBeanCopy extends CustomerAccessBean{
    private AddressAccessBean address=null;

    /* The following method overrides the getAddress method in * the CustomerAccessBean. */
    public AddressAccessBean getAddress(){
        if (address == null)
            address = super.getAddress();
        return address;
    }

    /**
     * The following method sets the address to the copy.
     */
    public void _setAddress(AddressAccessBean aBean){
```
Continuing the CUSTOMER and ADDRESS example, the session bean finder method would instantiate a CustomerAccessBean for each row in the CUSTOMER table and an AddressAccessBean for each corresponding row in the ADDRESS table. Then, for each column in the ADDRESS table, it sets the attributes in the AddressAccessBean (street and city). For each column in the CUSTOMER table, it sets the attributes in the CustomerAccessBean (name, age and address).

**Arbitrary SQL case**

In this case, there is a set of arbitrary SQL statements that perform database intensive operations. For example, the operation to sum all the rows in a table would be considered a database intensive operation. It is possible that not all of the selected rows correspond to an entity bean in the persistent model.

An example that could result in the creation of an arbitrary SQL statement is when a customer tries to browse through a very large set of data. For example, if the customer wanted to examine all of the fasteners in an online hardware store, or all of the dresses in an online clothing store. This creates a very large result set, but out of this result set, it is most likely that only a few fields from each row are required. That is, the customer may only initially be presented with a summary showing the item name, picture and price.

The way to solve this is by creating a session bean helper method. Depending on the requirements, the session bean helper method would either perform a read or a write operation. When performing a read operation, the method will return a read-only value object that is used for display purposes.

With proper data modeling, the number of cases of arbitrary SQL statements can usually be minimized.

Refer to 7.3.1, “Lightweight Object Query System” on page 141 for information about a tool to aid in the creation of read-only session beans and corresponding data beans for the arbitrary SQL case.

**8.3.8 Managing concurrent updates**

A concurrent update is when multiple processes, or multiple threads within the same process, attempt to update the same row of a table. There are situations in which concurrent updates may be allowed, but in most situations concurrent updates are definitely not desired.
In order to prevent two threads from concurrently updating a row, the first thread accessing the row must fetch the row using the \textit{for update} option. When the \textit{for update} option is used, an exclusive write lock is obtained for the row. With this write lock applied to the row, any application that attempts to access the row using the \textit{for update} is blocked.

If your application permits concurrent updates, it can just fetch the data, without locking the row.

Consider the \texttt{OrderProcess} scenario in which \texttt{UpdateInventory} needs to find all the products included in an order and update the inventory accordingly. Since the same products may be included in many other orders, find for update should be used, and it should be used as early as possible within a transaction scope to reduce the possibility of deadlocks. Therefore, the \texttt{UpdateInventory} algorithm may be represented by the following pseudo code:

1. Find all the products in an order, using “for update”
2. For each product, do:
   a. Deduct quantity from inventory
   b. Commit the change

Even if this process runs for a long time, any other process attempting to retrieve the same rows using “for update” will wait until the \texttt{UpdateInventory} algorithm is done.

8.3.9 Flush remote method

Since WebSphere Application Server does not write changes made on the entity beans to the database until the transaction commit time, the database may get temporarily out of synchronization with the data cached in the entity bean’s container.

A remote method named \texttt{flush} is provided that writes all the committed changes made in all transactions to the database. That is, the method takes information from the enterprise bean cache and updates the database. This method is defined in the \texttt{com.ibm.commerce.base.helpers.BaseJDBCHelper} class. This remote method can be called by a command. Use this method, only when absolutely required, since it is expensive in terms of overhead resources, and therefore, has a negative impact on performance.

Consider a logon command that has the code shown in Example 8-7.

\textit{Example 8-7   Example of using commitCopyHelper}

\begin{verbatim}
UserAccessBean uab = ...;
    uab.setRegisteredTimestamp(currentTimestamp);
\end{verbatim}
Until the transaction has been committed, the REGISTRATIONUPDATE column in the USERS table will not be updated with the current time stamp. The update only occurs at transaction commit time. In other words: the commitCopyHelper method does not actually commit the data to the database until the entire transaction is committed. In WebSphere Commerce, this happens with the current controller command returns from the performExecute method without an exception being thrown.

The flush method has to be used so that any direct JDBC query run in the same transaction, for example, a session bean that executes an SQL query using the JDBC API, returns the same data that has been committed via CMP EJBs.

### 8.3.10 Generating primary keys

In the WebSphere Commerce programming model, the persistence layer includes entity beans that interact with the database. As such, database records may be created when an entity bean is instantiated. Therefore, the ejbCreate method for the instantiation of an entity bean may need to include logic to generate a primary key for new records.

When an application requires information from the database, it indirectly uses the entity beans by instantiating the bean's corresponding access bean and then getting or setting various fields. An access bean is instantiated for a particular record in a database (for example, for a particular user profile) and it uses the primary key to select the correct information from the database.

The ejbCreate method is used to instantiate new instances of an entity bean. This method is automatically generated but the generated method only includes logic to set the primary key to a value specified by the caller.

You may need to ensure that the primary key is a new, unique value. In this case, you may have an ejbCreate method similar to code shown in Example 8-8.

**Example 8-8  Generating a unique primary key**

```java
import com.ibm.commerce.key.ECKeyManager;
...

class EntityBean {
    public void ejbCreate(int argMyOtherValue)
        throws javax.ejb.CreateException
    {
        //Initialize CMP fields
        MyKeyValue = ECKeyManager.singleton().getNextKey("table_name");
    }
```

Chapter 8. Developing new sites  157
MyOtherValue = argMyOtherValue;
}

In the code shown in Example 8-8, the getNextKey method generates a unique integer for the primary key. The table_name input parameter for the method must be an exact match to the TABLENAME value that is defined in the KEYS table. Be sure to match the characters and case exactly.

In addition to including the preceding code in your ejbCreate method, you must also create an entry in the KEYS table. Example 8-9 shows a sample SQL statement to make the entry in the KEYS table.

Example 8-9 Adding a table_name to the KEYS table

```
insert into KEYS (TABLENAME, COUNTER, KEYS_ID)
values ("table_name", 0, 1)
```

Note that with the SQL statement in Example 8-9, default values for the remaining columns in the KEYS table are accepted. Refer to the database schema documentation in the WebSphere Commerce Information Center for details about the semantics for the remaining columns. The value for the COUNTER column indicates the value at which the count should start. The value for KEYS_ID should be any positive value, not conflicting with an existing row.

If your primary key is defined as a long data type (BIGINT or NUMBER(38, 0)), use the getNextKeyAsLong method instead of the getNextKey method.

8.4 Best practices for JSP programming

Display pages return a response to a client. Display pages should be implemented as JSP pages. Although it is possible to generate the view output directly from a servlet, it is strongly discouraged.

In order to support multiple device types, a URL access to a view command should use the view name, not the name of the actual JSP file.

The main rationale behind this level of indirection is that the JSP page represents a view. The ability to select the appropriate view (for example, based on locale, device type, or other data in the request context) is highly desirable, especially since a single request often has multiple possible views. Consider the example of two shoppers requesting the home page of a store, where a shopper using a typical web browser and the other using a cellular phone. Clearly, the same home page should not be displayed to each shopper. It is the web controller's responsibility to accept the request, then based upon information about the user
determine the exact JSP template that should be executed to generate the shopper’s view.

The following are a list of the best practices described in this section for JSP development with WebSphere Commerce V5.6:

- Use JSTL in place of Java code
- Use WebSphere Commerce activation tags
- Use WebSphere Commerce maps to get request parameters
- Use StoreErrorDataBean for error handling
- Use an appropriate includes mechanism
- Use the JSPF file extension for JSP segments
- Use the escapeXml attribute to preserve HTML formatting
- Ensure XHTML compliance
- Use CSS to separate layout from content
- Add id attributes to links and form elements
- Invoking controller commands from within a JSP page
- Use the post method for form submission

### 8.4.1 Use JSTL in place of Java code

The JavaServer Pages Standard Tag Library (JSTL) is a collection of JSP tags that provide standard functionality most commonly sought by JSP page authors. JSTL has support for conditions, iteration, locale-sensitive formatting, and so forth. It also has an expression language that allows page authors to control how data is retrieved and displayed.

Store JSP pages should contain little or no Java code in favor of JSTL. Any business logic should be delegated to page-friendly data beans, and the remaining presentation logic should be implemented in JSTL.

There are two versions of JSTL:

1. Expression language based JSTL
2. Request-time expression based JSTL

For WebSphere Commerce, the former, expression language based JSTL, should be used.

JSTL 1.0 consists of four functional areas, which are exposed as separate tag libraries with their own name spaces:

- Core
- XML processing
- I18N (internationalization) capable formatting
- Relational database access (SQL)
Since the WebSphere Commerce programming model requires that data for JSP pages is retrieved through the use of data beans, the SQL library tags may not be used. Refer to 8.4.2, “Use WebSphere Commerce activation tags” on page 161 for information about using WebSphere Commerce specific template library tags for activating WebSphere Commerce data beans.

The use of JSTL will be shown in the following examples.

**Example 1: Retrieve product price**
This example assumes that a product data bean is available to the JSP page and uses it to display the minimum item price of the product if it is available and the string Not available otherwise.

Using Java, this task can be accomplished by means of direct access to the data bean and object methods as shown in Example 8-10.

```
Example 8-10   Java code in JSP for retrieving the product price
<% if (productDataBean.getMinimumItemPrice() != null) { %>
  <%= productDataBean.getMinimumItemPrice() %>
<% } else { %>
  <%=infashiontext.getString("NO_PRICE_AVAILABLE")%>
<% } %>
```

Using JSTL, the same task can be accomplished by means of the `<c:out>` and `<fmt:message>` tags as shown in Example 8-11.

```
Example 8-11   JSTL tags for retrieving the product price
<c:out value="${productDataBean.minimumItemPrice}"
  <fmt:message key="NO_PRICE_AVAILABLE" bundle="${storeText}" />
</c:out>
```

The `<c:out>` tag outputs the result of evaluating its value parameter or, if that is null, the value provided in its body. The `<fmt:message>` tag retrieves the locale-specific version of a message from a resource bundle.

**Example 2: Display product names and prices**
This example assumes that a category data bean is available to the JSP page and uses it to display the name and price for each product in the category.

Using Java, this task can be accomplished by means of direct access to the data bean and object methods in conjunction with the Java control facilities as shown in Example 8-12.
Example 8-12  Java code in JSP for displaying product names and prices

```<% ProductDataBean[] products = categoryDataBean.getProducts(); for (int i=0; i < products.length; i++) {   <%= products[i].getDescription().getName() %>
   <% if (products[i].getMinimumItemPrice() != null) { %>
   <%= products[i].getMinimumItemPrice() %>
   <% } else { %>
   <%= infashiontext.getString("NO_PRICE_AVAILABLE")%>
   <% } %>
%>
```

Using JSTL, the same task can be accomplished by means of the `<c:forEach>`, `<c:out>`, and `<fmt:message>` tags in conjunction with the JSTL expression language facilities as shown in Example 8-13.

Example 8-13  JSTL tags for displaying product names and prices

```<c:forEach var="product" items="${categoryDataBean.products}"
   <c:out value="${product.description.name}" />
   <c:out value="${product.minimumItemPrice}"
       <fmt:message key="NO_PRICE_AVAILABLE" bundle="${storeText}" />
   </c:out>
</c:forEach>
```

**Note:** The JSTL examples (Example 8-11 and Example 8-13) are more compact, easier to understand and easier to maintain than their Java counterparts (Example 8-10 and Example 8-12).

8.4.2 Use WebSphere Commerce activation tags

WebSphere Commerce data beans require activation prior to their use. For this purpose, a WebSphere Commerce-specific version, `<wcbase:useBean>`, of the JSP `useBean` tag is provided. This tag performs data bean activation in a
Java-free manner and is the recommended method of data bean activation in store JSP pages.

Using the WebSphere Commerce activation tags not only result in more easily maintainable code, as the following example will show, it is also a requirement for using the JSP previews environment.

**Important:** Data bean activation can fail and cause an exception condition. To comply with the JavaServer Page specification requirement that no output be written to the output stream when forwarding to other Web assets. Error pages and data beans should be activated at the beginning of the JSP page.

As an example, Example 8-14 shows the generic way of activating a category data bean.

**Example 8-14  Activating a data bean using DataBeanManager**

```jsp
<jsp:useBean id="dbCat"
    class="com.ibm.commerce.catalog.beans.CategoryDataBean">
  <% com.ibm.commerce.beans.DataBeanManager.activate(
      dbCat, request, response); %>
</jsp:useBean>
```

The recommended way of accomplishing the same task in WebSphere Commerce V5.6 store pages is shown in Example 8-15.

**Example 8-15  Activating a data bean using the wcbase:useBean tag**

```jsp
<wcbase:useBean id="dbCat"
    classname="com.ibm.commerce.catalog.beans.CategoryDataBean" />
```

**Exception:** The `DataBeanManager` class used in Example 3-13 provides two methods for activating data beans:

- `activate`
- `silentActivate`

A WebSphere Commerce-specific JSP tag is only provided for the former method. In some instances, the `silentActivate` method is required, and in such cases, the generic method will have to be used.
8.4.3 Use WebSphere Commerce maps to get request parameters

WebSphere Commerce provides the following customized versions of the implicit JSP objects `param` and `paramValues` to facilitate access to decrypted HTTP request parameters. These are outlined in Table 8-3.

Table 8-3  WebSphere Commerce-specific parameter objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCParam</td>
<td>A Map instance that maps the name of a request parameter to its single String value.</td>
</tr>
<tr>
<td>WCParamValues</td>
<td>A Map instance that maps the name of a request parameter to a String array of all values for that parameter.</td>
</tr>
</tbody>
</table>

Although, in many cases, the standard implicit objects `param` and `paramValues` provide equivalent functionality to their WebSphere Commerce-specific counterparts, the former are not guaranteed to work properly in the case of encrypted parameters, and hence `WCParam` and `WCParamValues` should be consistently used in store JSP pages.

Example 8-16 shows how to output the value of the `catalogId` request parameter. This approach will work even if the request parameters are being encrypted.

Example 8-16  Using WCParam to access the catalog ID from the request

```jsp
<c:out value='${WCParam.catalogId}' />
```

**Exception:** The WebSphere Commerce specific maps cannot be used to access additional parameters that are passed into a dynamically included page (that is, a page included by means of the `<jsp:include>` or the `<c:import>` tag). Thus, in the following example, SomePage.jsp and SomeOtherPage.jsp will not be able to access `showSubCategory` through `WCParam` or `WCParamValues` and will have to do so through `param` or `paramValues` instead:

```jsp
<jsp:include page="SomePage.jsp" flush="true">
    <jsp:param name="showSubCategory" value="true" />
</jsp:include>

<c:import url="SomeOtherPage.jsp">
    <c:param name="showSubCategory" value="false" />
</c:import>
```
8.4.4 Use StoreErrorDataBean for error handling

To display store-specific error messages in JSP pages, use the StoreErrorDataBean data bean. This data bean provides getter methods to do the following:

- Retrieve the store error key, `<ECMessageKey>.<Error Code>`
- Retrieve the error message parameters, that is, the substitution parameters used in the error messages

In mapping error codes to error messages, the StoreErrorDataBean data bean relies on the existence of the store error message properties file, named `storeErrorMessages_<locale>.properties`, where `<locale>` is the current locale (for example en_US).

The file should reside in the following directory:

- `WAS_installdir\installedApps\installedApps\<node_name>\WC_<instance_name>.ear\Stores.war\WEB-INF\classes\<store_dir>`
- `WCToolkit_installdir\workspace\Stores\Web Content\WEB-INF\classes\<storedir>`

Where:

- `<storedir>` is the name of the store directory.
- `<node_name>` is the name of the WebSphere Application Server node.
- `<instance_name>` is the name of the WebSphere Commerce instance.

The store error message properties file should contain error message definitions, such as the ones shown in Example 8-17 on page 164.

*Example 8-17  Sample store error messages*

```java
# UserRegistrationAddForm.jsp
_ERR_CMD_INVALID_PARAM.2020 = Type an e-mail address in the E-mail address field.
_ERR_CMD_INVALID_PARAM.2050 = Type your password in the Password field.
# UserRegistrationUpdateForm.jsp
_ERR_AUTHENTICATION_MINIMUMLENGTH_PASSWORD.2200 = The password you have entered contains fewer than {0} characters. Ensure that your password is at least {0} characters in length, including {1} digit(s) and {2} letter(s), and try again.
```

Example 8-18 shows an example of using the StoreErrorDataBean.

*Example 8-18  Using the StoreErrorBean to retrieve error information*

```xml
<wcbase:useBean id="storeError"
classname="com.ibm.commerce.common.beans.StoreErrorDataBean" scope="page">
<c:set target="$ {storeError}" property="resourceBundleName"
```
Error handling for JSP pages can be performed in various ways:

- Error handling from within the page
- Error JSP page at the page level
- Error JSP page at the application level

**Error handling from within the page**
For JSP pages that require more intricate error handling and recovery, a page can be written to directly handle errors from the data bean. The JSP page can either catch exceptions thrown by the data bean or it can check for error codes set within each data bean, depending on how the data bean was activated. The JSP page can then take an appropriate recovery action based on the error received. Note that a JSP page can use any combination of the following error handling scopes.

**Error JSP page at the page level**
A JSP page can specify its own default error JSP page from an exception occurring within it through the JSP error tag. This enables a JSP page to specify its own handling of an error. A JSP page that does not contain a JSP error tag will have an error fall through to the application-level error JSP page. In the page-level error JSP page, it must call the JSP helper class (com.ibm.server.JSPHelper) to roll back the current transaction.

**Error JSP page at the application level**
An application under WebSphere can specify a default error JSP page when an exception from within any of its servlets or JSP pages occurs. The application-level error JSP page can be used as a mall level or store level (for a single store model) error handler. In the application-level error JSP page, a call must be made to the servlet helper class to roll back the current transaction. This is because the Web controller will not be in the execution path to roll back the transaction. Whenever possible, you should rely on the preceding two types of JSP error handling. Use the application-level error handling strategy only when required.

### 8.4.5 Use an appropriate includes mechanism

JSPs support two includes mechanisms:
Static includes

The static include directive causes the content of the specified file to be textually inserted into the including file at compile time, that is, when the JSP page is translated into a Java servlet. JSP elements of the included file are thus effectively visible anywhere in the resulting JSP page and must have unique names to avoid name clashes. Typical Web containers are unable to detect changes made to the included file at run time, and hence the including file needs to be updated for the changes to take effect. The syntax for the static include directive is as follows:

```jsp
<%@ include file="filename.jspf" %>
```

Dynamic includes

The dynamic include actions include the response generated by executing the specified page during the request processing phase, that is, when the including page is requested by the user. Since what is included is the response generated by the page, and not the content of the page itself, scripting variables declared in the included file are not visible elsewhere in the resulting JSP page. Changes made to the included file, on the other hand, are transparent to the including file.

Two alternatives exist for the dynamic include:

- `<c:import url="filename.jsp" />`
- `<jsp:include page="filename.jsp" flush="true|false" />`

Unlike `jsp:include`, the `c:import` action provides a mechanism to access resources that can be specified through a URL, thus allowing page authors to get access to resources that reside outside the Web application. On the other hand, it lacks the ability to flush the response. Finally, `c:import` is comparatively more heavyweight and is therefore not appropriate when a lightweight solution is sought.

Whereas the dynamic include is more flexible than the static include, it falls short in terms of efficiency. As a consequence, use static includes whenever possible to avoid performance issues.

8.4.6 Use the JSPF file extension for JSP segments

JSP segments are separate JSP files that are intended to be included into another page as described in 8.4.5, “Use an appropriate includes mechanism” on page 165. There JSP segments need not be legal JSP pages and thus may not compile properly.

To enable code development and support tools, such as the WebSphere Commerce JSP batch compiler, to differentiate between the two types of files,
use the JSP extension for the source files of complete JSP pages and use the JSPF extension for the source files of JSP segments.

8.4.7 Use the escapeXml attribute to preserve HTML formatting

By default, the value of the escapeXml attribute of the JSTL \(<c:out>\) tag is true. This default behavior ensures that HTML special characters, such as \(<\), \(>\), \&, ', or ", contained in output strings are converted into their corresponding character entity codes and displayed properly in the HTML page produced by the JSP page. In some common WebSphere Commerce store scenarios, however, this behavior is unwanted. One such scenario is the display of prices in globalized stores. Envision a JSP containing the tag following tag:

\(<c:out value="${product.listPrice}" />\)

If, for instance, the price in the example is 3,544 Japanese yen, WebSphere Commerce will generate the following string in the underlying call to the getListPrice method of the ProductDataBean:

\(&yen; 3,544\)

The ampersand will then be escaped by the \(c:out\) tag, and the following will be written to the browser:

\(&amp;yen; 3,544\)

This will then be interpreted by the browser, and the following text is displayed to the user:

\(&yen; 3,544\)

To prevent the conversion, escapeXml should be explicitly set to false as follows:

\(<c:out value="${product.listPrice}" escapeXml="false" />\)

Another common scenario is the display of user-supplied text, such as product descriptions or dynamic text messages. If you expect advanced users to use HTML tags for formatting, such as line breaks or bulleted lists, set the escapeXml attribute of the \(c:out\) tag used to display the text to false as in the following example:

\(<c:out value="${productPromotion.longDescriptionString}" escapeXml="false" />\)

8.4.8 Ensure XHTML compliance

JSTL and HTML elements of your JSP pages should conform to the W3C XHTML standard and recommendations. Especially consider the following:

- Use lowercase for all element and attribute names. For example, \(<table>\), and not \(<TABLE>\).
- Enclose all attribute values in double quotation marks. For example:
  ```html
  <img value="car.jpg" width="20" />
  ```
  Not:
  ```html
  <img value='car.jpg' width=20/>
  ```

- Ensure that every element has an end tag or is self-terminating if it is empty. For example:
  ```html
  <p>Welcome to WebSphere Commerce!</p>
  ```
  or:
  ```html
  <c:set var="xyz" value="uvw" />
  ```

- For HTML-compatibility, include a space in all empty element tags before closing the tag. For example, `<br />` instead of `<br/>`.

- Ensure that the HTML page that your JSP page produces begins with a valid document type declaration. For example:
  ```html
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
  ```

To ensure XHTML-compliance, use XHTML validators, such as the WebSphere Studio Application Developer HTML validator or the W3C Markup Validation Service, found at the following address:

```
http://validator.w3.org/
```

### 8.4.9 Use CSS to separate layout from content

While not being strictly WebSphere Commerce specific, a general best practice for any web application is to separate the layout of the site from the content structure.

This can be accomplished using Cascading Style Sheets (CSS). The advantages of using CSS include:

- A less complicated HTML structure, resulting in simpler JSP files, which means shorter initial development cycle and easier code maintenance.
- As the HTML structure becomes structured for content, instead of layout, the site will be more accessible for other browser types (for example, text-to-speech readers for people with disabilities).
- The site can use the same HTML for several media (for example, a web browser, printer, a pervasive device, and just change the style sheet).
- The HTML documents are typically shorter, resulting in a marginally faster page load.
- Separating content from style is just plain logic.
For more information about using CSS, see “Related publications” on page 245.

8.4.10 Add id attributes to links and form elements

In order to support automated testing, we suggest that id attributes are added to all links, forms and form elements. Refer to 7.3.3, “CommerceQA test tool” on page 142 for an introduction to an environment for automated testing a WebSphere Commerce site.

A typical JSP code fragment to generate a category link would look like this:

```html
<a href="CategoryDisplay?langId=<%=languageId%>&\&storeId=<%=storeId%>&\&catalogId=<%=catalogId%>&\&categoryId=<%=categoryId%>">
 <%=categoryName%> </a>
```

This could be changed to the following, provided that it is can ensured that all category names are valid as XHTML identifiers:

```html
<a id="catlink_<%=categoryName%>" href="CategoryDisplay?langId=<%=languageId%>&\&storeId=<%=storeId%>&\&catalogId=<%=catalogId%>&\&categoryId=<%=categoryId%>">
 <%=categoryName%> </a>
```

Alternatively, the category ID could have been used for generation of the id attribute. The id value should be selected such that it can be used in a test script that should not be constantly updated to reflect changing id values.

Although not strictly required by the various testing tools, it is good practice also add id attributes to forms and their elements.

Note that all values of id attributes must be unique within the same XHTML document.

8.4.11 Invoking controller commands from within a JSP page

Although invoking controller commands from within a JSP page is not consistent with separating logic from display, you may encounter a situation in which this is required. If so, the ControllerCommandInvokerDataBean should be used.

Using this data bean, you can specify the interface name of the command to be invoked, or you can directly set the command name to be invoked. You can also set the request properties for the command.

When this data bean is activated by the data bean manager, the controller command is executed and the response properties are available to the JSP page.
If you do call the `setRequestProperties` method before activating this data bean, the parameters from the request object will be passed to the bean and, hence, also passed to the controller command. If you do, however, call the `setRequestProperties` method before activating this data bean, only the specified properties (those passed into the `setRequestProperties` method), as well as any default properties specified in the `PROPERTIES` column of the `CMDREG` table for the invoked command will be made available to the command.

Once the controller command has executed, you can execute the view.

You should not reuse the same instance of the data bean to invoke other controller commands, as it will contain data and state information from its original usage.

Refer to the WebSphere Commerce Information Center for more information about using the `ControllerCommandInvokerDataBean`.

### 8.4.12 Use the post method for form submission

Data can be sent from client to server by way of an HTML form using either the get or the post method. Servlets can handle either method properly by appropriately parsing and decoding the incoming request. In general, use of the post method is recommended for the following reasons:

- Get request data is transmitted as part of the URL string and is hence vulnerable from security viewpoint, unless the URL string is encoded. In contrast, post request data is sent as part of the HTTP message body and is hence considerably less exposed.

- Different platforms and Web servers can impose a limit on the length of the URL string, which includes the get request data.

- Get request data is restricted to the ASCII character set. In contrast, the post method, with the content type of multipart/form-data, can cover the entire ISO10646 character set.

### 8.5 Command best practices

The following sections describes best practices for creating new customized task and controller commands.

The following topics will be discussed:

- New command naming and hierarchy
- Using constants to hold class and method name
- Command error handling
8.5.1 New command naming and hierarchy

When creating a new controller or task command, there are a number of considerations to take.

**Class hierarchy**
Create both an interface and a class, each extending from the base interface and class, respectively. The new class must implement the new interface, as illustrated in Figure 8-3.

![Class hierarchy for new commands](image)

**Naming**
New controller commands should be named using a noun-verb combination, where the verb is in infinitive form. Table 8-4 gives some examples of bad controller command naming and better alternatives.

New task commands typically use a verb-noun combination instead.

**Table 8-4 Controller command naming examples**

<table>
<thead>
<tr>
<th>Bad command names</th>
<th>Good command names</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCommentCmdImpl</td>
<td>CommentCreateCmdImpl</td>
</tr>
<tr>
<td>CommentCreationCmdImpl</td>
<td></td>
</tr>
<tr>
<td>NewCommentCmdImpl</td>
<td></td>
</tr>
</tbody>
</table>
8.5.2 Using constants to hold class and method name

The WebSphere Commerce utilities for error logging and tracing require information about the current class and method names. The class name can be retrieved during run-time in Java using the following method:

```java
class().getName()
```

The advantage of this is that the developer does not have to worry about the actual class and package name when developing. Unfortunately, for debugging purposes, this also hides information. As result of this is the name of the current object class, not the name of the class in which the method is implemented.

For example, invoking `SubClass.main` from Example 8-19 will result in the following output:

```
Inside SubClass.method1()
Inside SubClass.method1()
```

While this is correct in the sense that `method1` is invoked on an object of the `SubClass`, for debugging purposes, the following output would be desired:

```
Inside BaseClass.method1()
Inside SubClass.method1()
```

In addition, as Java does not have a mechanism for retrieving the name of the current method, it is necessary to use either constants or literal strings in the logging or tracing statements.

**Example 8-19 Example of the drawback of using getClass().getName() for debugging**

```java
public class BaseClass {
    public void method1() {
        System.out.println("Inside "+getClass().getName()+.method1()";}

    public static void main(String[] args) {
        BaseClass baeClass = new BaseClass();
        baseClass.method1();
    }
}

public class SubClass extends BaseClass {
    public void method1() {
        super.method1();
```
In order to overcome these issues, it is recommended to add a constant to the class itself, containing the class name, as well as a constant string to each method with the name of the method, as illustrated in Example 8-20 on page 173. Notice that the entry and exit trace string can simply be copied between methods without changing the content. Example 8-21 on page 175 shows how to use these constants when throwing exceptions.

Example 8-20  Using constants for class and method names

```java
package mypackage;
import com.ibm.commerce.command.*;
import com.ibm.commerce.command.ras.*;

public class MyControllerCommand
    extends ControllerCommand
{
    /** use a local constant to avoid long tracing lines */
    private static final long COMPONENT = ECTraceIdentifiers.COMPONENT_EXTERN;
    /** save the class name in case this class is subclassed */
    private static final String CLASS_NAME = "mypackage.MyControllerCommand";

    public void performExecute()
        throws ECException
    {
        /** define the method name once */
        final String METHOD_NAME = "performExecute";
        ECTrace.entry(COMPONENT, CLASS_NAME, METHOD_NAME);
        ...
        ECTrace.trace(COMPONENT, CLASS_NAME, METHOD_NAME, "Trace string");
        ...
        ECTrace.exit(COMPONENT, CLASS_NAME, METHOD_NAME);
    }

    public void validateParameters()
        throws ECException
    {
    }
```

{  
/** define the method name once */ 
final String METHOD_NAME = “validateParameters“;
ECTrace.entry(COMPONENT, CLASS_NAME, METHOD_NAME);

...

ECTrace.trace(COMPONENT, CLASS_NAME, METHOD_NAME, “Trace string“);

...

ECTrace.exit(COMPONENT, CLASS_NAME, METHOD_NAME);
}

8.5.3 Command error handling

WebSphere Commerce uses a well-defined command error handling framework that is simple to use in customized code. By design, the framework handles errors in a manner that supports multicultural stores. The following sections describe the types of exceptions that a command can throw, how the exceptions are handled, how message text is stored and used, how the exceptions are logged, and how to use the provided framework in your own commands.

Types of exceptions

A command can throw one of the following exceptions:

- ECAppllicationException

  Throwing an ECAppllicationException is a signal that an application-level error condition has occurred. This exception is typically thrown if the error is related to the user. For example, when a user enters an invalid parameter, an ECAppllicationException is thrown. When this exception is thrown, the Web controller does not retry the command, even if it is specified as a retrievable command.

- ECSystsystemException

  This exception is thrown if a run-time exception or a WebSphere Commerce configuration error is detected. Examples of this type of exception include null-pointer exceptions and transaction rollback exceptions. When this type of exception is thrown, the Web controller retries the command if the command is retrievable and the exception was caused by either a database deadlock or database rollback.
Both of the above listed exceptions are classes that extend from the class
com.ibm.commerce.exception.ECException.

Exception handling is tightly integrated with the logging system. When a system
exception is thrown, it is automatically logged.

**Note:** New in WebSphere Commerce V5.6 is the StoreErrorDataBean which
uses a new properties file to map error codes to messages. Using the
StoreErrorDataBean is the recommended way to retrieve error messages
within a JSP page.

Refer to 8.4.4, “Use StoreErrorDataBean for error handling” on page 164 for
details about using the StoreErrorDataBean.

**Throwing predefined exceptions**

When creating new commands, it is important to include appropriate exception
handling. You can take advantage of the error handling and messaging
framework provided in WebSphere Commerce, by specifying the required
information when catching an exception.

Writing your own exception handling logic, involves the following steps:

1. Catching the exceptions in your command that require special processing.
2. Constructing either an ECApplicationException or ECSystemException, based
   upon the type of exception caught and the context of the error condition.
3. If the ECApplicationException uses a new message, defining the message in
   a new properties file.

Example 8-21 illustrates the first two steps. Note that the constants CLASS_NAME
and METHOD_NAME are defined using the method described in 8.5.2, “Using
constants to hold class and method name” on page 172.

**Example 8-21   Throwing an ECSystemException**

```java
try {
    // your business logic
}
catch(FinderException e) {
    throw new ECSystemException(
        ECMessages.ERR_FINDER_EXCEPTION,
        CLASS_NAME, METHOD_NAME,
        ECMessagesHelper.generateMsgParms(e.toString()),
        e);
}
```
The definition of the constant object ECMessage._ERR_FINDER_EXCEPTION is shown in Example 8-22.

**Example 8-22  Definition of ECMessage._ERR_FINDER_EXCEPTION**

```java
public static final ECMessage _ERR_FINDER_EXCEPTION =
    new ECMessage(
        ECMensajeSeverity.ERROR,
        ECMensajeType.SYSTEM,
        ECMensajeKey._ERR_FINDER_EXCEPTION);
```

The constant ECMensajeKey._ERR_FINDER_EXCEPTION is defined as the string _ERR_FINDER_EXCEPTION and refers to the key in the corresponding language-dependent properties file, ecServerMessages_<locale>.properties. The message for the locale en_US is shown in Example 8-23. The fourth parameter specified in Example 8-21 is substituted for {0} before the message is logged.

**Example 8-23  Message text for _ERR_FINDER_EXCEPTION for locale en_US**

_ERR_FINDER_EXCEPTION = The following Finder Exception occurred during processing: "{0}".

_ERR_FINDER_EXCEPTION is an example of one of the predefined messages, corresponding to standard exceptions.

**Table 8-5  Commonly used predefined system messages**

<table>
<thead>
<tr>
<th>Message object name</th>
<th>Exception</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ERR_FINDER_EXCEPTION</td>
<td>FinderException</td>
<td>The following Finder Exception occurred during processing: &quot;{0}&quot;.</td>
</tr>
<tr>
<td>_ERR_REMOTE_EXCEPTION</td>
<td>RemoteException</td>
<td>The following Remote Exception occurred during processing: &quot;{0}&quot;.</td>
</tr>
<tr>
<td>_ERR_CREATE_EXCEPTION</td>
<td>CreateException</td>
<td>The following Create Exception occurred during processing: &quot;{0}&quot;.</td>
</tr>
<tr>
<td>_ERR_NAMING_EXCEPTION</td>
<td>NamingException</td>
<td>The following Naming Exception occurred during processing: &quot;{0}&quot;.</td>
</tr>
<tr>
<td>_ERR_NULL_EXCEPTION</td>
<td>NullPointerException</td>
<td>A Null pointer exception occurred during processing.</td>
</tr>
<tr>
<td>_ERR_REMOVE_EXCEPTION</td>
<td>RemoveException</td>
<td>The following Remove Exception occurred during processing: &quot;{0}&quot;.</td>
</tr>
</tbody>
</table>
Implementing customized messages

If an exception condition that does not correspond to a predefined exception message occurs, it is possible to add a customized message for this condition.

Also, it is recommended that you add your own messages for your specific situations and document these messages in the operational guides. This can be a great aid in the daily site maintenance.

The steps involved in creating a custom messages are:

1. Create a new message file, `ecCustomMessages.properties` file, adding the new message text to this file. The custom message file must be placed in a directory that is in the application’s class path. An example is shown in Example 8-24 on page 177.

   **Example 8-24 Custom message file, ecCustomMessages.properties**

   _ERR_MY_ERROR = A custom error occurred. Additional details are "{0}" and "{1}"

2. Create a new interface to hold the message key constants for the custom messages. Example 8-25 shows an example of a new interface.

   **Example 8-25 Custom interface for message key constants**

   ```java
   public interface MyMessageKey {
       public static final String _ERR_MY_ERROR = "_ERR_MY_ERROR";
   }
   ```

---

<table>
<thead>
<tr>
<th>Message object name</th>
<th>Exception</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ERR_NUMBER_FORMAT_EXCEPTION</td>
<td>NumberFormatException</td>
<td>The following Number Format Exception occurred during processing: &quot;{0}&quot;.</td>
</tr>
<tr>
<td>_ERR_IO_EXCEPTION</td>
<td>IOException</td>
<td>The following IO Exception occurred during processing: &quot;{0}&quot;.</td>
</tr>
<tr>
<td>_ERR_GENERIC</td>
<td>Any other unexpected system error</td>
<td>The following error occurred during processing: {0}.</td>
</tr>
</tbody>
</table>

**Important:** Never modify any of the existing message properties files. Instead, create a new file as described. Although the actual file name can be chosen arbitrarily (see Example 8-26 on page 178), it is recommended to use the name `ecCustomMessages.properties`. This will make the site more standardized which can ease the introduction of new developers to the team.
3. Create a class, or interface, containing a constant instance of the `ECMessage` class. You can make the new class a subclass of the `ECMessage` class, thereby having access to the predefined messages through this new class. The disadvantage of doing so is that it is not apparent from the exception throwing code whether the message being reported is a standard WebSphere Commerce, or if it is a custom message. Example 8-26 shows an example of creating the custom message interface.

**Important:** When instantiating the custom message, ensure that you specify the name of the custom message file, created in the first step, as the last parameter. Also, the message type must be `ECMessageType.USER`, not `ECMessageType.SYSTEM`.

**Example 8-26  Custom interface for holding custom message instances**

```java
public interface MyMessages {

    public static final String MY_RESOURCE_BUNDLE = "ecCustomMessages";

    public static final ECMessage _ERR_MY_ERROR =
        new ECMessage(
            ECMensajeSeverity.ERROR,
            ECMensajeType.USER,
            MyMessageKey._ERR_MY_ERROR,
            MY_RESOURCE_BUNDLE);
}
```

When the new message has been defined, the message can be used in an exception as shown in Example 8-27. This will result in a message with the following description:

A custom error occurred. Additional details are "info 1" and "info 2"

**Example 8-27  Throwing an exception with a custom message**

```java
throw new ECApplicationException(
    MyMessages._ERR_MY_ERROR,
    CLASS_NAME, METHOD_NAME,
    ECMensajeHelper.generateMsgParms("info 1", "info 2"),
    <errorTaskName>, <properties_for_error_view>);
```

### 8.5.4 Tracing

Tracing can be crucial to determine problems with code running in a production environment, so a general recommendation for custom code is to trace early and often. Particularly, follow the these guidelines for tracing:
Entry and exit points to non-trivial methods should be marked as shown in Example 8-20 on page 173. If the method contains exception handling, the exit trace statement should be placed in the finally block.

Use trace statements to reflect the overall flow if the method contains conditional statements.

If the method has a catch block, this block should have a trace statement, logging the stack trace, unless this is logged by other methods (by throwing an ECEException, for example). This is done to avoid error masking. See Example 8-28 on page 179 for an example of this (note that Example 8-28 assumes that the instructions in 8.5.2, “Using constants to hold class and method name” on page 172 have been followed).

Trace the content of relevant state variables and parameters.

**Note:** Always use the tracing framework for outputting status messages. *Never* use `System.out.println` to output tracing information. The examples in this chapter should show how simple it is to use the tracing commands. Also, using `System.out.println` can have a severe impact on performance.

```java
Example 8-28   Tracing previously masked exceptions

try {
    ...
} catch (Exception x) {
    // this should not happen
    if (ECTrace.traceEnabled(COMPONENT)) {
        ByteArrayOutputStream bosTrace = new ByteArrayOutputStream();
        PrintStream psTrace = new PrintStream(bosTrace);
        x.printStack(psTrace);
        ECTrace.trace(
            COMPONENT, CLASS_NAME, METHOD_NAME,
            new String(bosTrace.toByteArray()));
    }
}
```

## 8.6 General security considerations

This section describes the following security related best practices:

- Access control
- Encrypt sensitive information
- Protect sensitive configuration files
- Protect sensitive parameters
Avoid default credentials and port numbers

8.6.1 Access control

All commands, URLs and other resources must be protected using the WebSphere Commerce access control framework. Any custom command that uses protected resources, such as data beans, must declare these using the getResources method.

Refer to the WebSphere Commerce Information Center for details about configuring the access control for your custom resources.

8.6.2 Encrypt sensitive information

Sensitive information, such as passwords and credit card information, must always be protected. In the database, this is done by encrypting all such data before storing it.

The class com.ibm.commerce.util.nc_crypt can be used to encrypt and decrypt such information. Example 8-29 shows an example of how to encrypt and decrypt values using the merchant key.

Example 8-29  Encrypting and decrypting values in a controller command

```java
package mypackage;

import com.ibm.commerce.exception.ECException;
import com.ibm.commerce.exception.ECSystemException;
import com.ibm.commerce.ras.ECMessage;
import com.ibm.commerce.ras.ECMessageHelper;
import com.ibm.commerce.ras.ECTrace;
import com.ibm.commerce.ras.ECTraceIdentifiers;
import com.ibm.commerce.server.WcsApp;
import com.ibm.commerce.util.nc_crypt;

public class MyCmd {
    private static final String CLASS_NAME = "mypackage.MyCmd";
    private static final long COMPONENT = ECTraceIdentifiers.COMPONENT_EXTERN;

    private String strMerchantKey = null;

    private String getMerchantKey() throws ECException {
        final String METHOD_NAME = "getMerchantKey";
        ECTrace.entry(COMPARTMENT, CLASS_NAME, METHOD_NAME);
```
try {
    if (strMerchantKey == null) {
        String _strMerchantKey = WcsApp.configProperties.getMerchantKey();
        strMerchantKey = nc_crypt.decrypt(_strMerchantKey, null);

        if (strMerchantKey == null) {
            throw new ECSystemException(
                ECMensaje._ERR_GENERIC,
                CLASS_NAME, METHOD_NAME,
                ECMensajeHelper.generateMsgParms(
                    "Unable to retrieve merchant key. nc_crypt.decrypt returned null!");
        }
    }
}

} catch (Exception x) {
    throw new ECSystemException(
        ECMensaje._ERR_GENERIC,
        CLASS_NAME, METHOD_NAME,
        ECMensajeHelper.generateMsgParms(
            "Unable to retrieve merchant key: "+x.getMessage(), x);
}

finally {
    ECTrace.exit(COMPONENT, CLASS_NAME, METHOD_NAME);
}

return strMerchantKey;

private String encrypt(String secret)
    throws ECException
{
    return nc_crypt.encrypt(secret, getMerchantKey());
}

private String decrypt(String secret)
    throws ECException
{
    return nc_crypt.decrypt(secret, getMerchantKey());
}

8.6.3 Protect sensitive configuration files

Any WebSphere Commerce installation will have several configuration files containing sensitive information. As described in “wcs_encrypt” on page 130, this information is obfuscated to avoid a casual onlooker to retrieve this type of information. This type of obfuscating should in no way be confused with industrial
strength encryption. If an attacker gains access to the merchant key in the obfuscated form, such as it is available from the instance XML, that attacker will be able to convert that into the clear-text value within minutes. Example 8-30 shows an excerpt from an instance XML file, revealing the obfuscated merchant key and site administrator password.

Example 8-30 Sample excerpt from an instance XML file

...  

InstanceName="demo"
InstanceType="local"
MerchantKey="wJNBcU/5MODw0bRLTuRtcvC7xiUN2sAc"
PDIEncrypt="off"
PVCHeader="false"
ParentResellerOrgDN="o=reseller organization,o=root organization"
SiteAdminID="wcsadmin"
SiteAdminPassword="afacWLqg1tr1bNupQsppiw=="
StoreWebModuleName="Stores"
SupportedLanguages="-1,
UrlMappingFile="C:\WebSphere\CommerceServer56\xml\mapping\urlmapper.xml"

...

It is therefore important to use all available mechanisms for limiting access to the files on the systems running the site, whether it be the web server, the application server or the database.

Side note: The clear-text values for the merchant key and the site administrator password, as shown in Example 8-30 are deadbeef12345678 and password, respectively.

8.6.4 Protect sensitive parameters

If sensitive information is passed to a URL command as a parameter, this must be protected. The first step is to ensure that the request is encrypted. This is done by setting the value in the HTTPS column in the URLREG and VIEWREG database tables to 1.

The second step is to mark the parameter as a sensitive parameter. This ensures that the tracing subsystem of WebSphere Commerce does not output the value of the parameter to the trace file if tracing is enabled for this feature.

Protecting a parameter is done by editing the instance XML file, which is located in the following directory:
Locate the ProtectedParameters node and add your parameter to the list. Example 8-31 shows the default settings.

**Example 8-31  Protecting parameters from being put into log files**

```xml
<ProtectedParameters>
  <Parameter display="false" name="logonPassword"/>
  <Parameter display="false" name="logonPasswordVerify"/>
  <Parameter display="false" name="logonPasswordOld"/>
  <Parameter display="false" name="password"/>
  <Parameter display="false" name="cardNumber"/>
</ProtectedParameters>
```

Adding your parameter to this list ensures that the tracing commands will output that parameter's value “starred out”, as in the following example:

```
storeId=10001&langId=-1&logonId=joe&logonPassword=******
```

### 8.6.5 Avoid default credentials and port numbers

From a security standpoint it is generally a good idea to change as many TCP/IP server port numbers as possible from their default values. While this does not secure the application as such, it will make it more difficult for a casual intruder to utilize known vulnerabilities on remote servers.

Also, refrain from using standard user names and passwords, such as wcsadmin, dbadmin or db2admin.
Migrating existing sites

Most businesses treat investments in technology updates and migrates as overhead expenses. You may understand perfectly well that you need to upgrade or migrate to the latest WebSphere Commerce version to be on the leading edge of the technological offering. However, providing a business justification to do so requires outlining the business benefits, a clear strategy and a meticulous plan.

Each business works with a unique set of requirements and challenges. As such, it is unlikely that there is a methodology that would fit the requirements for businesses. For instance, you may have a highly federated system with a load balancer, a system with lots of back-end integration, a system with highly customized databases, and so on. However, there are a number of best practices which could serve as a guide through these migrations.

This chapter provides you with an array of best practices, proven to contribute towards the successful migrations of WebSphere Commerce sites. After reading this chapter you should be able to chart a draft plan for your site’s migration needs by completing the Planning and Sizing Template (see 9.5, “Planning and sizing templates” on page 211). This will ascertain the milestones on which you need to focus as well as provide a rough estimation of time frame you may require to migrate your commerce site.
9.1 Why migrate?

This section deals with strategic reasons and business justifications for migration, as well as strategies which may be adopted for your WebSphere Commerce site migration.

Migrating your WebSphere Commerce system requires time and resource but in most circumstance the investment is worth it. After migrating your site, you will be able to use a host of new and powerful features and enhancements.

9.1.1 Business benefits of new products

The latest version of WebSphere Commerce provides a number of up-to-date IBM technologies and conforms to the current industry standards. For example, WebSphere Commerce V5.6 provides you with the following features:

- WebSphere Application Server V5.
  
  WebSphere Application Server uses a very convenient and powerful Web based interface. It no longer requires a separate database or a Web server to be installed and operational. Many significant improvements can help you to get your job done more effectively.

- WebSphere Studio Application Developer V5.1.1, based WebSphere Commerce development environment.
  
  Many existing customers have upgraded in order to be able to take advantage of the Eclipse based WebSphere Studio Application Developer environment. Now, the WebSphere Test Environment uses the full base WebSphere Application Server, allowing for a more precise testing experience.

- DB2 Universal Database V8.1.5.
  
  DB2 Universal Database V8.1.5 contains a number of maintainability, usability, performance, and scalability related enhancements.

- JavaServer Pages Standard Tag Library (JSTL) 1.0.
  
  The starter store JSP pages now provide the benefits of JSTL, including ease of customizations.

9.1.2 Business benefits of new capabilities

The capabilities made available in WebSphere Commerce evolve through customer and market centric development cycles. Collaboration is a key influencer in WebSphere Commerce development. In specific cases, partners in development influence plans for the coming years. A new version of WebSphere...
Commerce provides opportunities to take advantage of the new capabilities. This enables you to stay in tune with the marketplace and ahead of your competitors.

Consider adding the new features and functionality that are available in WebSphere Commerce V5.6 to increase revenue and improve the customer experience. For example, many of the marketing, advertising and convenience features are now built into WebSphere Commerce V5.6 just to do that.

The following sections highlight some key built-in features that you can exploit to enhance customer experience and drive business benefits. For a complete list of new features, refer to the "What's new in this release" topic in the WebSphere Commerce Information Center.

**Increased performance**
You may be satisfied with your existing WebSphere features and functionality and you may be interested in moving your existing implementation to the current version without any changes. Even in this case, you may still benefit from a site migration if the load on the application server and database components can be reduced.

In addition to page based caching techniques available in the past, fragment caching is also available, which allows fragments of the JSP file to be cached. You should realize performance gains with the new caching capabilities.

**Catalog and product management**
The main new features within the category of catalog and product management include:

- **Different navigational catalogs:**
  WebSphere Commerce has introduced the concept of a master catalog and a navigational (or sales) catalog. Use navigational catalogs to easily create a different catalog navigation experience for your customers.
  Additionally, these catalogs can be managed from the provided Web-based administrative tools without the need for further customization.

- **Catalog Filtering:**
  In WebSphere Commerce Business Edition, you can use the Catalog Filter to exclude any of the products or categories in a master catalog that you do not want to sell at any of your stores.

**Campaigns and promotions**
New features in the merchandise marketing category include:

- **Promotions redesign:**
The promotions system has been completely redesigned. It now provides support for redemption limits and promotion codes. Any promotion type may be distributed as a coupon. The promotions system removes all dependencies on the Blaze Advisor rule processing system yet maintains full backward compatibility with Blaze Advisor for those sites that have existing Blaze licenses.

- Coupons:
  With the addition of the coupons feature, you can offer electronic coupons to your customers. Coupons can be collected by the customer in a coupon wallet until they expire or are redeemed with a qualifying purchase.

- Rules-based discounts:
  Rule-based discounts support various new discount models.

Collaboration
WebSphere Commerce supports two types of collaboration functionality. To use one, or both features, you must first install the supporting software associated with each type of collaboration.

- Collaborative workspaces (WebSphere Commerce Business Edition only):
  Collaborative workspaces support asynchronous communication by way of Lotus Quickplace.

- Customer care:
  Customer care provides real-time customer service support through a synchronous text interface using the Lotus Sametime server.

Orders and inventory management
WebSphere Commerce provides various new capabilities that includes the following:

- Split orders that are based on inventory (WebSphere Commerce Business Edition only).
- Multiple ShipTo addresses (WebSphere Commerce Business Edition only)
- Availability-to-promise (ATP) inventory management that provides the availability date for the order based on real-time inventory.
- Refund management and Return Merchandise Authorization (RMA)
- Various Request for Quotation (RFQ) enhancements are available (WebSphere Commerce Business Edition only).
Web visibility - optimize your site for search engines
Optimize your site for search engines by ensuring that your JSP pages can generate data that can be optimally consumed by search engines.

To optimize your Web pages so that they are given more weight by search engines when ranking search results, WebSphere Commerce provides the information, tools and enablement features in the latest release to remove any inhibitors. This optimally indexes dynamic pages at your site. Consider the following items for your migration project. For additional information, refer to the WebSphere Commerce Information Center:

- URL Mapping: Provides a full set of indexable pages, matching the dynamic pages that have traditionally not been indexed by search engines.
- Site Map capability: A set of tools and techniques for optimizing the indexed content for your site.
- Suggestions on WebSphere Commerce optimization of the page content.
- Plan to enhance WebSphere Commerce Analyzer with reports. Track the search engine performance and the conversion rate.

Easily host multiple merchant storefronts
An Extended Site, or Ex-Site, is a business-to-business Web site that provides a richly featured site where the selling organization can set up different storefronts to target each of their business customers. When a business customer enters the Ex-Site storefront, any discounts, product configurations, and special instructions particular to that customer are displayed.

All features of the advanced B2B direct starter store, such as approvals, and account management also apply in a Ex-Site. The distinguishing characteristic of an Ex-Site store is that a Seller can provide a customized storefront and catalog to each business partner or customer.

Advanced analytics through WebSphere Commerce Analyzer
WebSphere Commerce Analyzer now includes redesigned installation and configuration components, decreasing install time. It also includes many new capabilities including single sign-on (SSO), a restriction on viewing reports based on authorization, the Reporting Framework which replaces the Brio Broadcast Server, and integration with Tivoli Web Site Analyzer.

9.1.3 Prepare for end-of-service of your product
WebSphere Commerce products are supported for a three year service window. A direct migration path is available to the current release from any version released during this service window. If you would like to migrate to the latest
version from a version that was released prior to this service window, you will have to migrate to an intermediate release before migrating to the current release.

It is a good practice to migrate at least once during the three-year product service window. WebSphere Commerce ensures API compatibility for the service window of the product. As such, your migration effort will be greatly reduced.

9.2 Migration strategy considerations

Migration to WebSphere Commerce V5.6 can be a complex undertaking, depending on your existing system configuration. Regardless of the migration approach you choose, the complexity of your site design, the level of customization, and so on, the migration may require a significant investment in time and resources. This section presents some factors that you may need to consider while formulating your migration strategy. For example, it is often cost effective to include any pending site redesign or enhancement projects along with your migration effort.

Your existing site may be integral part of a number of business imperatives. It may be generating revenue, requiring maintenance, and requiring updates. Many other activities may be ongoing with it or due to it. It may have a topology with only one machine or many machines (upwards of ten or so). Your site may be based on a WebSphere Commerce sample store with just a different presentation layer or flow, or your site may have highly customized code and database schema. The effort of the migration depends on these factors and others.

Clearly identity your plan targets and plan to test each of the targets. Also consider a roll-back plan from each target. Refer to 9.4, “Migration planning” on page 203 for more details.

9.2.1 Getting help

Migration complexity has been consistently decreasing compared to the previous releases. Migration is a far more efficient and manageable task now than ever before.

Depending on the complexity of your site, your migration schedule, the skilled resource available (refer to 9.4, “Migration planning” on page 203) and the critical nature of your business, you may consider getting help from an IBM services team or business partner. Refer to 1.1, “Further information” on page 10 for more information about available options.
You can outsource your complete project. Alternately, you can seek assistance from one or more experts, to help with migration while you provide the rest of the resources for the project. The latter approach may be useful if you have skilled resources to assist the external experts. This approach has the benefit of having skilled resources helping you with the actual migration tasks. It also helps the rest of your team in answering their questions and assists with problem determination, if required. Either way, in your plan, ensure that you account for skills transfer from the experts to your team so that your team can function efficiently on the migrated system when the expert leaves.

You may also consider outsourcing performance and throughput testing to an IBM services team, as they have the automated tools and processes to provide assistance with quick turn-around.

Finally, consider getting education to get familiar with the new features and how they may impact or improve your day-to-day operations.

### 9.2.2 Migrating the different environments

There are three environments that you will need to migrate:

1. **Development environment:**
   
   The development environment refers to the machines you use to develop custom code. Development environment migration uses the “in-place” migration approach, where all the migration steps are done on your existing machine. It is recommended that you have a second development machine available to maintain your existing WebSphere Commerce site.

2. **Test and staging environments:**
   
   The test environment refers to the test servers used during site maintenance, as well as the staging environment used to test configuration changes, such as catalog updates, before they are applied to the production environment.

3. **Production environment:**
   
   Production environment refers to your existing live environment or site used to serve your end customers.

The latter two environments are typically denoted as *runtime* environments, in that they are similar with respect to the installed WebSphere Commerce software stack. The development environment contains a different software stack, requiring a different migration procedure.

### Migrating the development environment

Although there is no requirement to migrate the development environment first, this is the recommended approach. This approach is recommended since if you
have any custom code, you will need to migrate (or, at least compile) it using the upgraded development environment before you can deploy it on your migrated runtime system.

The WebSphere Test Environment available in the WebSphere Commerce Developer product is based on the full WebSphere Application Server functionality (base edition). Thus, the migrated development environment will also provide you with an excellent ability for testing the functionality of your migrated site before attempting a runtime migration.

Since there is typically more than one existing development system, you can proceed with a migration of one, while retaining the ability to use the remaining systems to maintain your existing environment.

**Migrating the runtime environments**

After migrating the development environment, migrate the test environment to allow you to test the deployment of your custom code. Migrating a test environment before attempting your production environment is highly recommended. This allows you to practice migration on the test system and surface any issues before attempting it on your live production environment.

Finally, the runtime environment can be migrated using one of the approaches described in 9.3, “Migration approach considerations” on page 198. Ascertain your migration approach so that you can plan for it at the next stage of the project.

**Consideration for cloned environments**

If you have a cloned environment, and you are not installing your migrated environment on a new set of machines, you may choose to take part of your site offline for migration. This assumes that you can afford to have your site running at lower performance or with lower redundancy, for example during an off-peak time. By migrating part of your current WebSphere Commerce environment while the rest of it is still online, you can significantly decrease your downtime to the range of the switchover approach discussed in 9.3.1, “Switchover migration” on page 199.

**9.2.3 Handling complexity**

Divide the migration project into smaller tasks to isolate the scope of activities to logical stages. This would help isolate and resolve any potential problems.

The following provides one mechanism to divide the migration tasks and then further sub-divide them into even smaller tasks. The underline theme in doing so is:
Core migration and custom code migration
The key here is to differentiate migration from new feature development. This compartmentalizes the risks, and the testing and troubleshooting efforts into smaller, and more manageable problem domains. For example, if you have customized shipping or promotions code, you can divide the migration of your existing site into a core migration task including database migration and instance configuration migration and custom code migration. Once you complete the core migration, you discover that your custom code is impacted by API deprecation. In this case, consider bypassing your custom code and just test the standard functionality. Once you are satisfied with the core migration, you can focus on the custom code migration.

Having divided the migration task into different environments, you can further break down migration tasks by logically grouping together related features.

Adding new features
It is likely that in addition to migration, you wish to exploit new features and functionality provided by the new version of WebSphere Commerce. Isolate the new feature development task from the site migration task by performing site migration first and successfully testing it.

Itemize all additional requirements you have and identify the requirements which are logically related and should be done together as a group. For example, you may consider implementing all your promotions customization together first, then all your order management customization, and so on.

9.2.4 Component view
At this level, you further divide the components identified above into more specific tasks.

For example, if you have customized shipping or promotions code, you can divide migration of your existing site into a core migration task. This includes database migration, instance configuration migration, and custom code migration. Once you are done with the core migration and you discover that your custom code is impacted by API deprecation, you may consider bypassing your custom code and just test the standard functionality. Once you are satisfied with the core migration, you can focus on custom code migration.
9.2.5 Minimizing customizations

WebSphere Commerce Commerce provides you with a very rich set of features. By using out-of-the-box features, some customizations done by customers may have been avoided by simply using the out-of-box functionality. In some cases, customized features can be replaced by out-of-the-box functionality during a migration project. By rebuilding stores based on one of the new starter store models, it is possible for you to take advantage of the new functionality rather than maintaining these features in the existing site. It is often possible to reduce the amount of customizations that you must manage. This translates directly into lower operating costs.

Consider a classroom educational course from IBM Education if you are not familiar with the new features available in your target WebSphere Commerce version. There are also a number of online, self-paced educational courses available as well.

9.2.6 Performance optimization with dynamic caching

You can redesign JavaServer Pages (JSPs) to use the dynamic caching page and fragment caching technology. You will notice performance improvement with either of the two options. This will also result in measurable improvements in throughput by reducing the load on the application server and database.

If you employ personalization, it is recommended that you isolate the personalized content within JSP fragments to ensure efficient operation of the site. Using fragment caching will likely provide a very substantial performance improvement. You may consider using the new starter stores as a model to understand how fragment caching works.

9.2.7 Scalability

To meet the growing needs of your business consider hardware upgrades as well as adding clustering to your environment. WebSphere Application Server Network Deployment provides a centralized mechanism to manage your all your clones from one machine.

9.2.8 Database cleanup

Database cleanup should be an integral part of your WebSphere Commerce database maintenance. Regular maintenance of your database reduces additional disk requirements and increases the performance of your system.

It is suggested that to improve the performance of database migration you clean up your current WebSphere Commerce database by clearing the following
database objects. You can remove unused or obsolete objects from your
database using the database cleanup utility `dbclean`.

- Logs (for example, user traffic log)
- Any abandoned shopping carts
- Old guest users

If you put off your database maintenance to when you actually migrate, it will add
substantially to your migration service window. If you are not cleaning up your
database on a regular basis, then consider small but regular overnight cleanup
everyday before scheduled go-live date.

If cleaning up your database will take an unacceptable amount of time, consider
migrating it as it is. The performance of database migration has substantially
improved. It may actually be faster to migrate obsolete data than to clean it.
However, if you take this approach, make sure to start regular clean up of your
database right after your site migration!

### 9.2.9 Basing development on starter stores

Sample starter stores, are provided to quickly get you up and running by
customizing them to meet your requirements. If you used a sample store to base
your site, or if your new site design is very similar to one of the sample stores,
consider starting with a sample store and then adding your customizations. Your
site would benefit from all the new improvements delivered and showcased in the
new starter stores.

Furthermore, it is much easily to remove or turn off a feature than to add it
yourself. Refer to 8.1, “Basing development on a starter store” on page 144 for
more information on this topic.

### 9.2.10 Store archive (SAR) migration

In WebSphere Commerce V5.6, the role of a Store Archive has been focused to
that of a deployment vehicle for the WebSphere Commerce sample stores. Thus,
there are no longer tools that update the contents of a SAR. Tools such as the
Store Profile editor, and the Tax and Shipping editor have been integrated with
the WebSphere Commerce Accelerator tool.

Changes to a store's content are made either during development before the
assets are packaged into a store archive (SAR), or as part of managing a
deployed store. Given that there is no need to manage the content in a SAR, the
Store Archive Publish process has been streamlined so that a new SAR is no
longer created based on a sample SAR. Instead, the parameter values that were
previously specified (for example, during the creation of a WebSphere
Commerce V5.4 SAR) can now be specified during the publishing process. The Store Archive Publish process has been enhanced so that it is controlled by XML descriptors within the SAR. This provides added flexibility and control to the store deployment. Also, note that each sample SAR shipped since WebSphere Commerce V5.5 now contains all supported languages.

If you are considering SAR migration, re-evaluate your need to do so unless you need to publish the migrated SAR many times. For more information about SAR migration, refer to the Migration Guide for your platform.

9.2.11 Delta migration

Delta migration refers to the scenario where you migrate your production database in two stages. First, you migrate a copy of your production database on, say, to your test environment. Once you are satisfied with the migration you take your production environment offline and simply migrate the delta changes in the production data while integrating the migrated delta with the previously migrated database.

This type of delta migration scenario is not supported by WebSphere Commerce since a complete end-to-end database migration should no longer take more than an overnight service window. Improvement in WebSphere Commerce database migration techniques allow you migrate large databases very quickly.

However, if there is some reason that you find yourself wanting delta migration then re-evaluate your need since dividing up your database migration purely due to its size may not be required.

Note that there may be some valid delta migration scenarios as well. For example, assume you have migrated a copy of your production database on your test environment. Now, you want to modify the data on your test environment and then propagate these changes to the production database after the production database has been migrated. On the new migrated site, you may want to present a completely redesigned master catalog or provide redesigned navigational catalogs or any other non-operational data. In such a case, you have two choices.

- Capture all the changes done to your migrated database in your test environment, in a script file, and execute the script after your production database migration.
- Alternately, set up a staging server and migrate a copy of your production database on it. Make any changes on this database as desired. Once you are satisfied with all the changes, bring your production environment offline, and migrate your production data. Finally, propagate all the non-operational data from your staging server.
9.2.12 How custom code impacts the process

Regardless of the approach you choose for the migration, you may need to recreate and test your custom code to account for changes in the new version. Several changes in the structure and deployment of custom code require your attention. Review the WebSphere Commerce Migration Guide for further information. Here we list a few key areas. Note, however, once you have migrated your development environment, WebSphere Studio Application Developer will indicate to you the custom java code that requires attention.

As an example, the following list highlights some of the most apparent API differences between WebSphere Commerce Suite V5.1 and WebSphere Commerce V5.6:

- String converters in access beans are no longer supported.
- Security roles are now required for Enterprise JavaBeans (EJBs).
- Fine grained access control methods have changed.
- Command parameter validation methods have changed.
- User registration and logon mechanisms have changed.
- Calculation framework customizations are now policy based.
- Access to order items through the OrderItemAccessBean has changed to now use a new Item bean class.

9.2.13 How database customizations impacts the process

If your database schema was highly customized and requires migration, consider extending the database migration scripts to migrate your custom tables as well.

The database migration script uses an XML plan file which controls the behavior of database migration utility. All the database changes are passed to the database migration tool as SQL files using this plan file. Thus, it allows you to control the database migration behavior and fine tune it for your requirements. Refer to the appendixes of the WebSphere Commerce Migration Guides to find further information about this.

9.2.14 Migrating your WebSphere Commerce instance or recreating it

The WebSphere Commerce instance migration (wcim) tool automates the migration and packaging of your application as an EAR file as well as its deployment to the target WebSphere Commerce system. This includes migrating your store assets, JDBC information, changing the Web server and database names, and so on. This tool runs in both the development environment (either
based on VisualAge for Java or WebSphere Studio Application Developer) and the runtime environment.

It is highly recommended that you use this tool to migrate your WebSphere Commerce instance. However, it is possible but not usually recommended to create a new instance and point it to the migrated database, as described in the appendixes section of the WebSphere Commerce Migration Guides.

9.3 Migration approach considerations

As mentioned in 9.2.2, “Migrating the different environments” on page 191, there are three environments to migrate. These can be divided into two types of environments:

- Development environment
  The development environment refers to the machines you use to develop custom code. Development environment migration uses the “in-place” migration approach, where all the migration steps are done on your existing machine.

- Runtime environments
  The runtime environment refers to your testing, staging and production environment where you deploy your custom code and serve your site.

There are three approaches to migrating the runtime environment:

- Switchover migration
  This approach involves creating a separate WebSphere Commerce V5.6 environment on a remote node, while keeping the previous WebSphere Commerce environment operational.

- Coexistence migration
  Coexistence migration involves migrating to a WebSphere Commerce V5.6 environment and creating a new instance while the previous WebSphere Commerce environment is still running on the same machine. This is possible if the existing WebSphere Commerce node has sufficient capacity to operate both of the instances (and software stacks, for a single node installation) at the same time.

  **Note:** This approach is not supported for Microsoft Windows platforms.

- In-place migration
The in-place migration scenario transforms the existing production runtime. This means that the production system has to be inoperational for the duration of the migration.

**Attention:** This migration scenario is not recommended, as it results in the highest downtime for the runtime environment. Additionally, this approach also involves a high risk, as a viable roll-back option is not readily available.

The approach that you use depends on the availability of new hardware, or the capacity of the existing hardware. The fastest and safest approach to migration is the switchover migration.

Table 9-1 shows a comparison between the different migration scenarios, who will be described in detail in the following sections.

**Table 9-1  Comparison of the migration scenarios**

<table>
<thead>
<tr>
<th>Category</th>
<th>Switchover</th>
<th>Coexistence</th>
<th>In-place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impact to Production</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Fall-back risk</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Additional hardware requirements</td>
<td>At least one additional runtime environment</td>
<td>No additional requirements</td>
<td>No additional requirements</td>
</tr>
</tbody>
</table>

### 9.3.1 Switchover migration

To use the switchover migration approach, you must set up duplicate hardware. You will set up your WebSphere Commerce V5.6 production environment on a separate set of machines, while your existing WebSphere Commerce production environment is still operational and unaffected by the migration.

This approach is the recommended scenario, but is only suitable when the necessary hardware to duplicate the current runtime environment is available to create a test environment where the migration takes place. After this test environment is fully migrated and tested, it becomes the new production environment. Then the previous production environment can be taken off line.

With this approach, the WebSphere Commerce server downtime is kept to a minimum. The migration is carried out and tested on the test environment while the production servers remain running in parallel. By the time the migration and testing are complete, the database from the migrated environment can be out of
sync with the production database. To reconcile the data, take the previous WebSphere Commerce production environment offline and re-migrate the database based on your experience with the previous migration. When the migration is complete, switch over the test environment as the production environment.

Turning the newly migrated environment into production is a relatively simple task. Figure 9-1 on page 200 shows a high-level layout of the switchover migration scenario approach implementation.

The following four steps are the major steps in a switchover migration. The numbers refer to Figure 9-1:

1. Migrate the development environment.
2. Install WebSphere Commerce V5.6 on additional hardware and migrate your existing site to this environment.
3. Take the existing production system offline and migrate your database.
4. Reroute Internet traffic to the new system.

The switchover migration approach offers clear benefits. The migration is comparatively low risk, because the impact to your existing production site is minimal. In the event of encountering problems during the migration, the original WebSphere Commerce production environment serves as the recovery machine.
9.3.2 Coexistence migration

The coexistence migration approach uses your existing test and production hardware. However, you must have sufficient spare capacity to concurrently run two versions of WebSphere Commerce. If you use this approach, you should have a test environment available. IBM iSeries and IBM zSeries systems often migrate by using the coexistence migration approach.

Figure 9-2  Overview of a coexistence migration

Generally, the steps for a coexistence and in-place migration are the same. The main difference is that you must select the coexistence options during installation and migration, when applicable.

Note: This migration approach is not supported on the Microsoft Windows or Linux platforms. Additional information about this migration approach can be found in the WebSphere Commerce Migration Guide.
9.3.3 In-place migration

The in-place migration approach uses your existing test and production hardware. Choose this migration approach if you do not have any duplicate hardware or extra resource on your existing production environment.

The test environment allows you to practice migration on a test system. You can resolve any issues before you migrate the production environment.

Because an in-place migration may impact your live production site, it is the riskiest migration approach. An in-place migration is not recommended, if you do not have a test environment. Downtime can be reduced considerably if your production environment is cloned or if it has built-in redundancy.

An in-place migration is performed on the production server. This approach typically has the longest server downtime in comparison to the other two approaches, although the entire migration does not have to be done all at one time. Migrating some components individually may help to minimize the production server downtime.

Consider this approach when there are limited hardware resources and only the production server is the available system for migration (such as single-node implementations). This assumes that high availability is not required and/or no additional systems are available. If this is the case, this approach offers an option to migrate to WebSphere Commerce V5.6.

Having a multiple-node environment may provide the chance to consolidate functionality and nodes. Freeing up some nodes can allow you to migrate them offline. The tradeoff is a reduction in the processing capacity, but the site still remains operational.

Detailed and careful planning is required for this approach. Although supported, it may not be preferred if there is server downtime. Working directly on the production server may be of concern.
The in-place migration involves three main steps. The numbers refer to Figure 9-3:

1. Migrate the development environment.
2. Install WebSphere Commerce V5.6 on an existing test environment, or set up a new one, then migrate your existing site to this environment.
3. Install WebSphere Commerce V5.6 on your existing production environment, and migrate your site to the V5.6 level. This step impacts your production site. Decide when to take down your production site based on your experience when migrating the test environment.

### 9.4 Migration planning

WebSphere Commerce integrates with several other software products to provide award-winning e-commerce solutions. Migrating WebSphere Commerce can be a complex and challenging task given the number of products and subsystems involved. A complete migration plan must include all the necessary
upgrades to the application code and WebSphere Commerce servers. The plan must also consider business requirements, while ensuring that during the migration, service cannot be significantly affected. That is, your existing business must continue during the migration.

Migrating to a new version of WebSphere Commerce may impact several aspects of your business. Your WebSphere applications are far more than just application code. Along with the application code, it is important to consider the following items:

- Custom application code.
- Development environment (including source control management systems).
- Build processes.
- Testing.
- Runtime environments.
- Deployment processes.
- Education and training.

Migrating WebSphere Commerce requires careful planning. Since the process involves a number of interdependent software products, it can take a considerable amount of time, and requires specialized skills. Careful migration planning will ensure minimal impact on your business.

In general, the migration effort is determined by:

- Your current version of WebSphere Commerce. The older the version, the greater the effort.
- The site architecture and configuration, before and after migration.
- The level of site customizations, before and after migration.
- Integration of additional software products.
- To some extent, the size of the database.

A WebSphere Commerce migration project typically requires the following six tasks:

- Inventory the existing WebSphere Commerce site assets.
- Itemize the requirements for the post-migrated site.
- Plan the migration activities.
- Prepare for migration.
- Migrate.
- Test migrated site.
The following sections provide greater detail on the above and provide you with checklists which you could use to devise your migration plan.

### 9.4.1 Planning activities

During the planning activities for migrating a WebSphere Commerce site you should consider evaluating the current site assets and how they currently meet your needs for e-commerce. Based on this information, you will be able to determine the skills required to complete the migration effort. The following sections do not provide a comprehensive list of assets, but simply the more common assets at most sites.

**Evaluating current site assets**

Take an inventory of your current WebSphere Commerce site assets, which include:

- **Topology information**, such as information about the following items in your existing site:
  - Machines
  - Instances
  - Cluster members
  - Databases

- **Custom code developed**:
  - Enterprise Java Beans and commands
  - Custom Java code
  - Java Server Pages
  - Database schema customizations
  - Code deployment scripts
  - Data loading scripts
  - WebSphere Commerce tools and so on.

- **Existing business processes**:
  
  To ensure that existing processes continue to work flawlessly, you first need to identify them. It may even be possible to better support these processes with the availability of new capabilities in the new version of WebSphere Commerce. For example, should free shipping be provided before or after tax?
Source code:
Ensure you have access to all custom source code as you may need to migrate any deprecated code.

Integration points:
- WebSphere family of products, for example MQ, LDAP, messaging and so on.
- Third-party software, for example Payments, taxation and so on.
- Back-end integration.

**Evaluating skilled resources required**
Inventory the skills of the people that will be involved in the migration effort. Consider if you have the appropriate database, Java, coding, testing, and IT systems administration skills to migrate your customizations and support your migration needs:

- Database administration skills, to help migrate custom database extensions, such as new tables or views, for example:
  - IBM DB2 Universal Database (UDB) or Oracle.
  - An understanding of relational database concepts.
  - The ability to perform basic SQL queries.

- WebSphere Commerce development skills, to migrate custom Java™ code.

- WebSphere Commerce administration skills, to minimize the impact to your existing production site.

- Software installation, configuration, and maintenance skills, to install and configure WebSphere Commerce and co-required software products, for example:
  - The operating system.
  - WebSphere Application Server administration.
  - IBM HTTP Server administration.
  - Understanding of TCP/IP, HTTP, and Secure Sockets Layer (SSL) protocols.

**9.4.2 Gathering migration requirements**

Itemize the requirements for the post-migrated site.

- New requirements: New requirements can include following examples as dictated by your reasons for migration (please refer to 9.2, “Migration strategy considerations” on page 190 for additional examples):
– Upgrading or adding hardware
– Using new features or functionality
– Replacing existing custom-code with out-of-the-box functionality
– Address additional performance or scalability requirements
– Accommodate new or changed Business Processes

▶ Software level support: If there are integration points (as listed in the previous section) for the new site then ensure that the third party vendor claims integration with the new version of WebSphere Commerce.

### 9.4.3 Planning the migration activities

The activities for a typical migration project are:

▶ Installing the WebSphere Commerce V5.6 software products, or upgrading to the WebSphere Commerce V5.6 software levels.

▶ Migrating the WebSphere Commerce instance configuration (and if required, the WebSphere Commerce Payments instance configuration) to the WebSphere Commerce V5.6 level.

▶ Migrating the database schema to the WebSphere Commerce V5.6 level. Depending on the extent of database customizations, you may have to migrate your database more than once on your test machine prior to migrating your production environment. For example, you may want to fine tune the database configuration parameters to minimize the database migration time.

▶ After you have migrated your site, you must test it to validate a successful migration.

Now is the time to complete the planning of the migration tasks. Use the following checklist of key items as a guide:

▶ Consider a phased migration approach as discussed in 9.2.3, “Handling complexity” on page 192.
  – Consider migrating your environments in this order:
    a. Development environment
    b. Test environment
    c. Production environment
  – For clustered or federated environments consider a piece-meal migration as discussed in “Consideration for cloned environments” on page 192.
  – First migrate existing applications on a new software stack, then include customizations.
Decide on your production migration approach. Refer to 9.2, “Migration strategy considerations” on page 190 for more details.

- Switchover.
- Coexistence.
- In-place.

Ensure that you have the required skills (or available as needed) depending on the customization requirement of your site.

Validate hardware and software requirements.

- Ensure the new required software stack is supported by IBM and by your internal corporate standards
- Ensure the planned hardware meets your production requirements. For example, consider additional disk requirements beyond what is mentioned in the WebSphere Commerce Installation Guide and in the WebSphere Commerce Migration Guide. Extra resources may be needed for backups.

Plan for education.

Ensure that you understand the new version of WebSphere Commerce, including the new features and changes which have occurred. To do so review the WebSphere Commerce V5.6 documentation or take a self-paced, on-line educational courses.

**Note:** If you do not understand features required by your site design then up-front education is better than learning on-demand. Learning during the implementation phase may have adverse effect on both the implementation as well as project plan.

Consider migrating a backup copy of your production database during your development or test environment migration.

Consider database cleanup, as discussed in 9.2.8, “Database cleanup” on page 194.

Consider utilization of caching technology and fragment caching.

Pay particular attention to integration points.

Plan for backups and have a fall-back plan should you encounter any difficulty during the migration.

Review the migration roadmap available in the WebSphere Commerce Migration Guide and prepare your project plan. Refer to 9.5, “Planning and sizing templates” on page 211 for a sample that you may use and enhance for your needs.
9.4.4 Preparing for migration

The following is a checklist of the tasks and activities that you must complete in preparation of the bulk of the migration effort:

- Study the WebSphere Commerce Migration Guide for your platform that provides detailed, step-by-step directions to lead you through the migration processes and tools. Additionally, you can also review the *Keeping Commerce Applications Updated, SG24-6320-00* redbook, which provides an example of an end-to-end migration to WebSphere Commerce V5.6 on the Windows platform.

- Do not let the migration impact the maintenance schedule to the current WebSphere Commerce site.
  - Identify one of the development systems for migration and ensure that the other systems are sufficient for your needs.
  - If only one machine is available, consider installing the new version of WebSphere Commerce and its software stack on the same machine. Perform an in-place install on the test machine without overwriting the current stack.

- Before you migrate, ensure you have a way of restoring the system to a known, operational state.

- Complete any premigration activities that may be required.

- Production environment migration.
  - Practice migration at a test environment before migrating the production environment.
  - If your database schema was highly customized and requires migration, consider extending data migration scripts to migrate your custom tables as well. This is discussed on 9.2, “Migration strategy considerations” on page 190.
  - Decide on switching method (hostname/IP switching, DNS update, or NAT).

- Prepare test plans.

9.4.5 Migrating the site

Ensure that you have a migration plan as discussed earlier. Follow the WebSphere Commerce Migration Guide as per your plan to complete migration.

If you discover that you require any repetitive manual steps, then consider creating a script for your subsequent migration attempts. For example, to make database updates. When considering these scripts, include the production
environment migration, where downtime is extremely critical and manual steps or errors in the process can slow down or be detrimental to the migration process.

If you have read the *WebSphere Commerce Migration Guide* and *Keeping Commerce Applications Updated, SG24-6320-00* and have the required resources, consider migrating your custom code and database concurrently to save some time.

### 9.4.6 Testing the migrated site

After you have migrated your site, you must test it to ensure that the site functionality still meets your expectations and that the site is performing well. Along with the migration of your site, you should also migrate test cases supported by your site. If you did not have test cases, consider creating them and account for them in your plan. Testing is a project target which gets affected by project delays. If the plan does not account for this phase and provide contingency, you run the risk of runtime disruptions to your live site if testing is not adequate.

Unit test your site to ensure each piece or unit of code works well. This type of testing should be done on an on-going basis as units of code become available.

Perform functional verification test (FVT) of your site to ensure it supports the business processes as expected. You can perform FVT on an on-going basis, once the development of a function is complete or at the end of the development cycle.

Perform system verification test (SVT) of your site to ensure it delivers adequate response times and sufficient throughput. Use the caching technologies and stress-test the site to ensure that it can handle the load over peak periods. If you lack skills and tools to test performance and throughput consider the ideas in 9.2.1, “Getting help” on page 190.

### 9.4.7 Going live

After you have migrated your development environment and test environment you are ready to move ahead with migrating the production environment.

Installation of WebSphere Commerce V5.6 and its required products can be done while the current site is operational. Only the database migration requires that the production site be taken offline.

To minimize the risk of down time due to unexpected database migration problems it is recommended that you migrate a copy of your production database
during your development environment migration or during your test environment migration.

9.5 Planning and sizing templates

Depending on your site, the migration process may take a few days or several weeks.

1. The first step, migrating the development environment, may take several weeks.

2. Next, migrating the test environment can take up to two weeks.

3. Finally, migrating the production environment can be scheduled to occur during a short service window. The downtime of your site is determined by the size of this service window.

There are a number of activities which go into migrating your site. Figure 9-4 shows a sample diagrammatic representation of the migration activities.

Table 9-2 shows a sample planning worksheet that you can use and enhance for your needs.

<table>
<thead>
<tr>
<th>Task</th>
<th>Sub Task</th>
<th>Time</th>
<th>Assumptions, Comments, Implementors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site analysis and preparation</td>
<td>Review the current site and gather new requirements.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Review the WebSphere Commerce Installation Guide and WebSphere Commerce Migration Guide. Access any referred technotes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Sub Task</td>
<td>Time</td>
<td>Assumptions, Comments, Implementors</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Development environment migration</td>
<td>Ensure you met all prerequisites.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Install and configure WebSphere Commerce Developer.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate your development environment instance configuration.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate your custom code.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate your development database.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test your migrated applications in the development environment.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate Payments.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test environment migration</td>
<td>Ensure you have met all the prerequisites.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Install and configure WebSphere Commerce on your test environment.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate you test environment instance configuration.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate your test database.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complete all post-instance and database migration tasks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate WebSphere Commerce Payments.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complete any additional migration steps.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implement dynamic caching.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Sub Task</td>
<td>Time</td>
<td>Assumptions, Comments, Implementors</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Production server migration</td>
<td>Ensure you have met all the prerequisites.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Install and configure WebSphere Commerce on your production environment.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate your production environment instance configuration.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate your production database.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complete all post-instance and database migration tasks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Migrate WebSphere Commerce Payments.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complete any additional migration steps.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Perform FVT and SVT to adequately test your migrated site</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Get education for the day to day maintenance</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post launch support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Project management</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contingency (for unknown or unexpected issues)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total project hours:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 10. Integration with existing systems

External systems integration is a key feature of a WebSphere Commerce solution. In WebSphere Commerce business logic is enabled for integration and built-in adapters and interfaces are provided for common integration points. Additionally, the WebSphere Commerce integration solution leverages integration standards by using various IBM connectivity solutions.
10.1 WebSphere Commerce integration capabilities

WebSphere Commerce can integrate with:

- Customers by multiple channels.
- Back-end systems within the enterprise.
- Partners and suppliers.

To help integrators easily connect with external systems, WebSphere Commerce provides reference applications. Reference applications, available from the WebSphere Commerce support page, contains documentation and sample code to integrate with a back-end system.

The messaging system gives WebSphere Commerce the ability to communicate with its external environment. This communication includes sending messages to and receiving messages from back-end systems or external systems, as well as sending notification to customers and administrators that events have occurred within WebSphere Commerce.

Figure 10-1 depicts how WebSphere Commerce can integrate with existing applications.

![WebSphere Commerce integration points](image)

For example, you can set up the messaging system to send e-mail messages notifying your customers that their orders have been shipped. You can configure WebSphere Commerce to send message to a back-end system whenever an order is created at your store. This order information can be used by the back-end system to do necessary order fulfillment processing. The back-end system can later send order status messages back to WebSphere Commerce.
indicating that order delivery has taken place, or an order invoice has been
issued. An e-mail can also be sent to update the customer.

Inbound messages are used to run commands in WebSphere Commerce based
on messages coming from back-end systems. Outbound messages can be
generated by the outbound messaging system in order to update back-end
systems with events that have taken place, such as a new customer order.

Figure 10-2 shows WebSphere Commerce components and utilities used to
communicate with external systems. For example, XML files can be used as
input by the Loader utility which uses the files to populate the WebSphere
Commerce database. The Extractor utility can be used to extract data from the
database into various file formats. XML messages are also used as input to the
Messaging system, based on the content of the XML messages the Messaging
system can perform business logic or send messages to an external system.

![WebSphere Commerce Integration capabilities](image)

**10.1.1 Messaging architecture**

The WebSphere Commerce messaging subsystem is comprised of two
component subsystems for inbound and outbound messaging. To speed
development, WebSphere Commerce provides many predefined inbound and
outbound messages. The predefined messages provide common messaging
functionality for fulfillment and back-end messaging needs. In addition,
WebSphere Commerce provides an architecture for adding new inbound or
outbound messages and transports.
As seen in Figure 10-3, the inbound messaging system can receive XML-based messages via the HTTP protocol and WebSphere MQ. The outbound message system provides a means of sending notifications via e-mail to customers and administrators, send XML-based messages via the WebSphere MQ and WebSphere Interchange Server transports, and write a message to a file.

### 10.1.2 Predefined messages

The messaging system is prepared to send and receive a number of pre-defined messages in XML format. This format offers a high degree of readability, making the messages easy to modify and maintain. You can also use the legacy message format. However, the XML message format is recommended. You can also add new messages. For new inbound messages, you can associate them with either existing WebSphere Commerce commands, or commands that you have created.

WebSphere Commerce provides several messages that the system can use to communicate with an external system. In addition to the messages provided you can also create custom messages. Refer to *WebSphere Commerce V5.5 Handbook book, Customization and Deployment Guide, SG24-6969-00* for further details on integration with WebSphere MQ.
10.2 Web services

This section briefly explains the Web services infrastructure in WebSphere Commerce and helps you understand the internal Web services architecture. It also describes the method to expose WebSphere Commerce business processes as Web services and how WebSphere Commerce can access Web services hosted by external systems.

This information is intended for those who want to expose the business processes in WebSphere Commerce as Web services and developers who need to understand how to enable and implement Web services for WebSphere Commerce.

10.2.1 Overview

Web services are a new breed of Web applications. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions that can be invoked ranging from a simple request to complicated business processes. Once a Web service is deployed and registered, other applications can discover and invoke the deployed service. The foundation for Web services are standards such as Simple Object Access Protocol (SOAP), the Web services Description Language (WSDL), and the Universal Description, Discovery, and Integration (UDDI) registry.

WebSphere Commerce includes code and documentation that allows you to enable WebSphere Commerce business functions as Web services. You can allow WebSphere Commerce to be the service provider by enabling its business functions as Web services that can be accessed by external systems. You can also allow WebSphere Commerce to be the service requestor by enabling it to invoke Web services that are hosted by external systems.

10.2.2 Why use Web services?

Web services allow applications to be integrated more rapidly, easily and less expensively than ever before. Integration occurs at a higher level in the protocol stack, based on messages entered more on service semantics and less on network protocol semantics, thus enabling loose integration of business functions. These characteristics are ideal for connecting business functions across the Web, both between multiple enterprises and within a single enterprise. They provide a unifying programming model so that application integration both inside and outside the enterprise can be done with a common approach, leveraging a common infrastructure. The integration and application of
Web services can be done in an incremental manner, using existing languages and platforms and by adopting existing applications.

A Web service is an interface that describes a collection of operations that are accessible through the network by using standardized XML messaging. A Web service is described using a standard, formal XML notation, called its service description. The service description includes all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. WebSphere Commerce is designed to support Web services, both as a provider of Web services and as a requestor (client) to other Web services provider.

### 10.2.3 Web services architecture

The Web services architecture describes three roles: service provider, service requester, and service broker; and three basic operations: publish, find, and bind. A network component can play any or all of these roles. The service providers publish Web services to a service broker. Service requestors find required Web services by using a service broker and bind to them.

In a typical scenario, a service provider hosts a network-accessible service module, which is an implementation of a Web service. The service provider defines a service description for the Web service and publishes it to a service requestor or a service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry hosted by the service broker. The service requestor uses the service description to bind with the service provider and invoke or interact with the Web service implementation.
This solution provides run-time support to enable business processes that are defined by WebSphere Commerce as Web services.

For further information on WebSphere Commerce back-end integration refer to the WebSphere Commerce Information Center at the following URL:

http://publib.boulder.ibm.com/infocenter/wc56help/index.jsp
Appendixes
Project team creation and common roles

There are a number of the common roles that are needed within a team to execute a WebSphere Commerce project. In this appendix, we outline the process of creating a team and briefly describe the key roles. This information can be used as a guide to assess what roles and responsibilities make sense and are appropriate for a given project context.

It is possible for one or more of these roles to be performed by a single person during a project.

Some of the roles will be performed over the complete duration of the project, while others will only apply to certain phases of the project development method.

Having identified the roles, we then show how these roles relate to the migration tools, indicating which tool would be used at a particular point in the project by specific role performers.
Creating the project team

Although apparently obvious, note that all projects require a team of dedicated and skilled people to perform the activities that will ensure the project's success. A project team should be formed by the project manager. When selecting available people and assigning them to roles, the project manager must consider such factors as:

- Skills and expertise.
- Project objectives.
- Time scale for deliverables.
- Customer requirements for new or replacement site functionality.
- Geographic location for overlapping projects and teams or sub-teams of specialists.
- Customer intention to utilize their own staff for development or support.
- Training.
- Multicultural support in the site.
- Internationalization and translation.
- Double-byte character set (DBCS) enabling.

**Note:** A project team may consist of both customer and consulting resources, which is generally the case when a project requires skills transfer between the consulting firm and the customer, or the customer has requested that it should have the necessary skills required to administer the site after the project has ended. It is important to realize that time or resources spent supporting or training customer personnel should be included as a separate part of project development estimates.

An ideal commerce project, specifically one based on WebSphere Commerce, requires a set of common roles for team members. These required, or more appropriately, recommended roles are described in detail in “Project roles” on page 230.

Identifying team members and then assigning each to a role, or possibly roles, can be performed in many different ways. IBM Software Services has found two strategies for forming project teams, in particular, to be helpful during project design, each of which are discussed in detail here:

- The core competency strategy.
- The holistic strategy.
The core competency strategy

This strategy assumes that several of the team members have the relevant skills required by the project, and it enables the project manager to divide the roles and responsibilities according to the existing skills or competencies.

For example, if some members of the team are skilled, or have been certified as experts on specific WebSphere Commerce subsystems, such as Order, Member, or Catalog systems, then the project manager should assign design work involving these subsystems to the available “experts.”

An overview of the core competency strategy is provided in Figure A-1.

![Core Competency Team Assignment Strategy](image)

The core competency strategy fosters a team structure that is based on technology assignments, which in turn, afford each team member the opportunity to hone and refine their skills. At the conclusion of the project, each team member is likely to be better prepared to assume more challenging design and development responsibilities on future engagements, such as commerce projects. In addition, project models based on this strategy generally produce greater specialization and expertise in specific commerce areas within a services organization, skills the organization can call upon and sell later.
The holistic strategy

The holistic strategy is based on the requirement of making each team member equally effective in performing all project tasks, hence the reason for its name.

An overview of the holistic strategy is provided in Figure A-2.

![Holistic Team Assignment Strategy](image)

**Figure A-2  Holistic team assignment strategy.**

The holistic strategy is very useful when creating a workforce that can adapt to a variety of projects or work assignments. This strategy enables team members to focus on developing a breadth of commerce knowledge prior to refining it in given competencies. For example, when this strategy is employed, the project manager can expect and count on any team member being capable of design, coding, and testing most project artifacts, such as WebSphere Commerce commands or JSPs. To succeed using this strategy, the project manager must ensure the team has, or is in the process of obtaining, skills and experience with all core Java technologies, such as JSPs, commands, and EJBs, as well as ensuring that the team understands and has experience with core project activities, such as site design, implementation, and testing.

Either strategy, or the combination of elements of each, may work for a particular project. Prior to selecting a team strategy, you are encouraged to assess the
nature and scope of your project early to help you determine how best to form your project team and manage them through a specific strategy. Even after having made a decision, you will want to reassess your team strategy between project phases, because you may need to pursue a new strategy for any given phase.

Note: Whenever practical, we recommend not making significant changes to your project team during the design phase. Each new member added to the team will require appropriate lead time for them to establish a bearing with regard to the project’s current course and direction. Although the resulting delay is rarely convenient, it is absolutely necessary if the new members are to be fully integrated into the team to become productive members.

In practice, resource exclusivity is a rarity on most projects. Therefore, project managers should comprehensively document all design tasks and work assignments and then post the documentation so that all team members may access it. Whenever possible, the project manager should use standard document templates and layouts to enable new and old team members to assimilate the information quickly. Combined, these activities will enable the project manager to mitigate some of the problems or issues that may arise from people movement during the design phase.

Guidance on selecting the strategy

Selecting the appropriate strategy for a project will necessarily reflect the circumstances of the project itself. However, a useful guide is to assess the skill level of team members and their previous experience with commerce projects.

If less than 20% of the team have previous commerce experience, it is advisable to build core commerce skills before commencing the project. While it may appear attractive to “build skills on the job,” this is likely to introduce significant scope for error in estimating the time to complete phases and may impact the project delivery date.
If less than 40% or 50% of the team have previous commerce experience, this would indicate that the core competency strategy would be appropriate for the project.

If more than 40% or 50% of the team have previous commerce experience, this would indicate that the holistic strategy would be appropriate for the project.

**Note:** The percentages suggested here are for guidance purposes only; each project should be assessed individually.

# Project roles

The most common WebSphere Commerce project roles, and the skills necessary to perform the roles, are summarized in Table A-1.

<table>
<thead>
<tr>
<th>Project role</th>
<th>Main phases</th>
<th>Product related skills required</th>
<th>Other skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project manager/lead</td>
<td>All</td>
<td>General understanding of WebSphere Commerce product and architecture.</td>
<td>Project management, preferably in the e-commerce area.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Manages and leads project team along all phases of the project and also acts as a contact point to interact with customer IT and test teams.</td>
</tr>
<tr>
<td>Technical lead/architect</td>
<td>All</td>
<td>WebSphere Commerce architecture and design skills.</td>
<td>Preferably, having experience in technically leading a project team during all phases of a project.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Looks after the overall project technical architecture/design, quality assurance of the solution, knowledge transfer to customer, and mentoring to the project technical team members.</td>
</tr>
</tbody>
</table>

*Table A-1  Common project roles and associated skills*
<table>
<thead>
<tr>
<th>Project role</th>
<th>Main phases</th>
<th>Product related skills required</th>
<th>Other skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSphere Commerce designer</td>
<td>Design</td>
<td>WebSphere Commerce design skills.</td>
<td>Technical design experience; OOD and implementation experience; knowledge of Java technology; knowledge of the development environment and tools.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Based upon project scope and complexity, one or more WebSphere Commerce designers can work to create detailed project technical designs. The work effort can be divided based on common WebSphere Commerce subsystems, such as Catalog, Member, Order, and communication with back-end or legacy systems.</td>
</tr>
<tr>
<td>Database creation or migration expert</td>
<td>Design</td>
<td>Understanding of WebSphere Commerce schema, and database creation or migration knowledge.</td>
<td>Database knowledge, such as DB2 and Oracle; SQL knowledge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The main database creation or migration effort should be complete by the end of design phase; it is assumed that the data model for the new or migrated site is available to the design team of the WebSphere Commerce architect and designers.</td>
</tr>
<tr>
<td>WebSphere Commerce programmer</td>
<td>Build and Site Test</td>
<td>WebSphere Commerce programming model.</td>
<td>J2EE programming, knowledge of the development environment and tools.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WebSphere Commerce architect/technical lead and WebSphere Commerce designers can also participate in building new or migrated sites.</td>
</tr>
<tr>
<td>System administrator</td>
<td>Design, Build, Site Test and Launch</td>
<td>WebSphere Commerce administration.</td>
<td>Team repository administration; setting and working with live or simulated back-end systems, for example, using MQ for connectivity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Responsible for all types of project software and hardware infrastructure setup and maintenance work.</td>
</tr>
<tr>
<td>Project role</td>
<td>Main phases</td>
<td>Product related skills required</td>
<td>Other skills</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>----------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Test lead</td>
<td>All</td>
<td>General understanding of WebSphere Commerce product and architecture.</td>
<td>Detailed understanding of functioning of customer site; experience in leading testing activity; test tool skills.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lead the test team during all site test phases; work with customer IT team to support SIT, SVT, UAT, and launch preparation phases.</td>
<td></td>
</tr>
<tr>
<td>Testers</td>
<td>FVT</td>
<td>Knowledge of the test tools used by the test team.</td>
<td>General understanding of Web-based systems and customer site information and behavior.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any members of the project team, who can satisfy the criteria may participate in this activity.</td>
<td></td>
</tr>
<tr>
<td>DBA</td>
<td>Design, Build, Site Test and Launch</td>
<td>Some knowledge of WebSphere Commerce administration.</td>
<td>Database administration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The database creation or migration activity role could also be performed by the DBA.</td>
<td></td>
</tr>
</tbody>
</table>
Analysis, site maintenance and enhancement phases

In this appendix, we provide a brief description of the analysis and site maintenance and enhancement phases. These represent the entry and exit points, respectively, for a typical WebSphere Commerce project.
Analysis phase

During the analysis phase, we recommend conducting a high-level fit-gap analysis to begin defining the project scope and to help develop a rough sizing of the work effort. A good vehicle for this activity is to participate in a fit-gap analysis workshop with the customer.

The output of this activity should provide information to support development of a plan to proceed with the project. In terms of a normal consulting engagement, this information leads to a statement of work (SoW) or a document of understanding (DoU). Figure B-1 provides an overview of this phase.

![Figure B-1](image_url)  

**Figure B-1  Overview of analysis phase and deliverables.**

The scope of the first SoW or DoU would typically be limited to cover only that work that is required to produce a valid site design, which is a major output of the design phase. A second SoW or DoU would then be used to describe the remaining tasks for the project that come after site design.

As an additional observation, when engaged with an in-house project, which is one that is owned and operated by the customer, it is likely that additional time should be allocated for obtaining project sponsorship.

A checklist summary for the activities and work products of the analysis phase is provided in Table B-1.
Table B-1  The analysis phase.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase name</td>
<td>Analysis phase.</td>
</tr>
<tr>
<td>Short description</td>
<td>This phase is conducted before a WebSphere Commerce project starts formally. The customer and the organization providing the project team must work together to define and document high-level objectives, project scope, and a work proposal ready for the WebSphere Commerce project to start.</td>
</tr>
<tr>
<td>Input work products</td>
<td>RFI, RFP, or customer business requirements document.</td>
</tr>
<tr>
<td>Output work products</td>
<td>As a result of conducting a fit-gap analysis workshop, several outcomes would normally result:</td>
</tr>
<tr>
<td></td>
<td>- Initial fit-gap analysis document containing rough sizing.</td>
</tr>
<tr>
<td></td>
<td>- Refined customer business requirements document.</td>
</tr>
<tr>
<td></td>
<td>- Education roadmap.</td>
</tr>
<tr>
<td></td>
<td>- Knowledge communicated face-to-face on WebSphere Commerce and its high-level relevance to the customer business requirements.</td>
</tr>
<tr>
<td></td>
<td>- First SoW or DoU.</td>
</tr>
</tbody>
</table>

If this is a migration project, a high-level analysis of the current site's assets and the current data model, will help in performing the fit-gap analysis during this phase.

Analysis phase sizing

In many projects, customers will require budget and planning numbers as soon as possible, often before detailed requirements have been established. Although the design work will not begin until after the analysis phase, it may be possible to provide an approximate sizing by comparing the project with similar scenarios or projects. These prior experiences may give an indication of the number of hours and resources required.

Analysis phase sizing is experience-based and is also referred to as business-process or scenario-based sizing. The approach taken is typically to break down the project as follows:

- Identify each high-level business process or key subsystem.
- For each of these processes or subsystems, identify the business functions that must be implemented.
For each business function, produce a list of functional requirements and implementation scenarios, and assign an assessment of expected effort based on prior experience.

Add the total assessed effort.

For any other project aspects not already included, add a percentage effort for coding or testing activities accordingly. The percentage used should again be based on prior experience.

This sizing approach does not require a design, and the result will certainly be less accurate than a sizing performed after design has been completed. However, it will help to provide the customer with an early indication of the likely project sizing, and so assist them with early activities, such as ordering equipment.

Site maintenance and enhancement phase

This phase marks the beginning of site activities that are related to maintaining and enhancing (based on new and changing business requirements) site assets after the successful launch of the customer e-commerce site. The project team completes its major role before this phase starts, and this phase would normally continue for the remaining life of the site.

A checklist summary for the activities and work products of the site maintenance and enhancement phase is provided in Table B-2 on page 236.

Table B-2  The site maintenance and enhancement phase.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase name</td>
<td>Site maintenance and enhancement phase.</td>
</tr>
<tr>
<td>Short description</td>
<td>This phase marks the beginning of site activities that relate to maintaining and enhancing site assets after the new or migrated site is launched. The project team completes its major role before this phase starts.</td>
</tr>
<tr>
<td>Input work products</td>
<td>Site launch phase outputs (see Figure 6-2 on page 107). A live e-commerce site following completion of the WebSphere Commerce project.</td>
</tr>
<tr>
<td>Output work products</td>
<td>Error-free live site. If new or changed business requirements have been fulfilled, the live site will have been successfully enhanced.</td>
</tr>
</tbody>
</table>

For a migration project, it is assumed that before the start of the project the customer site is already in the maintenance phase. After the migrated site has
been launched, however, the customer site will normally re-enter the maintenance phase to continue its life cycle.

After launching the site, the customer IT or support team can take charge of the ownership of maintaining and enhancing the site. Besides regular maintenance, this team may also have the mandate of adding new features or enhancing the existing features of the live site, based on new or changing business requirements.

In order for the customer IT or support team to manage this activity effectively, it must have the following:

- A fully functional development environment so that the site code may be fixed or enhanced as desired.
- A test environment on which to assess changes before moving the code to the production site.
Appendix C. Additional methods

This Redpaper uses specific terminology to describe a WebSphere Commerce development method. The terms are in common use among the Toronto-based IBM Software Services commerce team and originate from IBM Global Services terminology, subsequently adapted slightly for WebSphere Commerce implementation projects.

In this appendix, we consider other methods used within IBM or as part of engagements where IBM consultants work with customers on implementation projects. The purpose is to highlight that the literal terms used in this Redpaper are less significant than the concepts they embody, with the proviso that all consultants and developers working on a project must have an agreed upon and common understanding of the terms being used.
Deciding the method and terminology

In deciding the method and terminology to be used, the starting point should be to assess:

- Project team and customer team familiarity with existing methods and terms.
- The match between the conceptual model that underpins the method and the project requirements.

For example, a project team that has worked with the customer team on previous projects is likely to have developed their own local interpretation or “dialect” of terms that have evolved as an optimized amalgamation of standard methods and terms. It would be unwise to discard the procedural and teaming benefits of this localization, as long as the basic rigor of the method is maintained.

Similarly, a method that focuses on code implementation using an object-oriented model may be inappropriate to use on a project that simply requires reconfiguration of standard components, rather than development of code using an object-oriented language.

IBM methods

Not surprisingly, IBM has a well-established discipline for managing development projects. Within that context, there will be development methods that are optimized according to the nature of the project work. For example, the process of designing and developing a system performance assessment tool may have differences when compared with the design and development of an embedded hardware component for a Point-of-Sale terminal.

The intent is to ensure that a balance is found between a process that is insufficiently structured, and so allows freedom and flexibility, but is hard to measure and document, against a process that is too structured, and so constrains creativity and that may be bureaucratic or slow.

The Rational Unified Process (RUP)

The Rational Unified Process (RUP) is an software engineering process. Its purpose is to provide a disciplined approach to identifying and assigning tasks and responsibilities within a development project. RUP is intended to be configurable according to project circumstances.

The complexity of most significant development projects make it unlikely that a simple sequential development model will succeed; rather an iterative process
must be applied where earlier steps will be revisited and refined in the light of subsequent understanding and progress on the problems and tasks. The Rational Unified Process is an iterative process at a macro and micro level. It supports iterative development in a number of ways, for example, using frequent test releases of code or assets to allow customer testing, verification, and feedback.

At the macro design level, *Inception, Elaboration, Construction* and *Transition* can be identified in the process. These activities are basically periods of the initial planning, design, build and closing and moving on to the next project cycle. The next cycle will repeat these phases. At the micro level each activity may go through several iterations of itself. For example during a construction phase: coding, testing and re-coding may take place a number of times. Figure C-1 gives an overview of the Rational Unified Process.

![Figure C-1  Rational Unified Process overview](image)

Also shown in Figure C-1, the Rational Unified Process identifies a number of disciplines that are practiced during the various phases. These disciplines are practiced during all phases but the amount of activity in each phase varies. Clearly the requirements discipline will be more active during the earlier inception and elaboration phases for example.
Table C-1 shows how the RUP phases correspond approximately to the phases identified within this Redpaper.

**Table C-1 Comparison between RUP and Redpaper terminology**

<table>
<thead>
<tr>
<th>RUP</th>
<th>Redpaper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>Analysis, discussed in Appendix B, “Analysis, site maintenance and enhancement phases” on page 233.</td>
</tr>
<tr>
<td>Elaboration</td>
<td>Design discussed in Chapter 3, “Design phase” on page 27. Phase activities are:</td>
</tr>
<tr>
<td></td>
<td>- Solution outline</td>
</tr>
<tr>
<td></td>
<td>- Macro design</td>
</tr>
<tr>
<td></td>
<td>- Micro design</td>
</tr>
<tr>
<td>Construction</td>
<td>Build and test, discussed in Chapter 4, “Build phase” on page 59 and Chapter 5, “Site test phase” on page 89. Phases activities are:</td>
</tr>
<tr>
<td></td>
<td>- Project implementation.</td>
</tr>
<tr>
<td></td>
<td>- Functional Verification Test.</td>
</tr>
<tr>
<td></td>
<td>- System Integration Test.</td>
</tr>
<tr>
<td></td>
<td>- System Verification Test.</td>
</tr>
<tr>
<td></td>
<td>- User Acceptance Test.</td>
</tr>
<tr>
<td>Transition</td>
<td>Site launch, discussed in Chapter 6, “Launch phase” on page 105.</td>
</tr>
</tbody>
</table>

The Rational Unified Process maps disciplines to roles. There are many roles but the roles break down into 4 basic sets of roles: *Analysts, Developers, Testers, Managers*. Members of the team may take on more than one role. More than one team member may have the same role. Each role may require the practice of more than one discipline. Table C-2 shows the basic mappings between roles and disciplines.

**Table C-2 RUP disciplines (vertical) mapped to RUP role sets (horizontal)**

<table>
<thead>
<tr>
<th></th>
<th>Analysts</th>
<th>Developers</th>
<th>Testers</th>
<th>Managers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Modeling</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Rational Unified Process can be followed without using Rational Software, it is just a process specification after all. However, the Rational Unified Process provides specific guidance (called Tool Mentors) on how to use Rational Software when following the process. The disciplines identified in the Rational Unified Process such as requirements analysis, design or testing map to specific pieces of Rational software and artifacts that this software generates. The Rational Unified Process is a process that can be ‘bought into’ as much or as little as is required.

For more information on the Rational Unified Process visit the following URL:

http://www.ibm.com/software/awdtools/rup

**Summary**

If a match is found between a standard method and the project requirements, then clearly the method is a strong candidate for use on the project. If no clear match is found, then the closest approximations should be considered to assess the impact of the discrepancies. It may be that a close match can be adjusted sufficiently to support the project, while not compromising the architectural foundation of the method itself.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redpaper.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 246. Note that some of the documents referenced here may be available in softcopy only.

- *Keeping Commerce Applications Updated, SG24-6320-00*
- *WebSphere Commerce V5.4 Catalog Design and Content Management, SG24-6855-00*
- *WebSphere Commerce V5.5 Handbook book, Customization and Deployment Guide, SG24-6969-00*

Other publications

These publications are also relevant as further information sources:

- *WebSphere Commerce Installation Guide*
- *WebSphere Commerce Migration Guide*

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM Business Consulting Services
- IBM Global Services
- IBM Software Services for WebSphere
- WebSphere Commerce Information Center
http://publib.boulder.ibm.com/infocenter/wc56help/index.jsp

- **WebSphere Application Server V5 Information Center**
  http://publib.boulder.ibm.com/infocenter/wasinfo/index.jsp

- **IBM DB2 Universal Database Information Center**
  http://publib.boulder.ibm.com/infocenter/db2help/index.jsp

- **IBM Cloudscape Information Center**
  http://publib.boulder.ibm.com/infocenter/clscp10/index.jsp

- **W3C pages on cascading style sheets (CSS)**
  http://www.w3.org/Style/CSS/

- **A List Apart on CSS**
  http://www.alistapart.com/topics/css/

- **Caching WebSphere Commerce pages with the WebSphere Application Server V5 dynamic cache service**

- **The Extensible Stylesheet Language Family (XSL)**
  http://www.w3.org/Style/XSL/

- **Using the WebSphere Commerce V5.6 auto build and deployment tool**

- **WebSphere CommerceQA V5.6 automation package**

- **HttpUnit at SourceForge**
  http://httpunit.sourceforge.net/

- **JUnit at SourceForge**
  http://junit.sourceforge.net/

- **Rational Unified Process**
  http://www.ibm.com/software/awdtools/rup

### How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:
Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services
ibm.com/services
Best Practices and Tools for Creating WebSphere Commerce Sites

Systematic Project Development Method

This IBM Redpaper introduces a systematic project development method that was specifically adapted for the creation of e-commerce sites using the WebSphere Commerce family of software products. This Redpaper details two WebSphere Commerce development scenarios:

- Creating a new e-commerce site.
- Migrating from existing WebSphere Commerce.

Development and Migration Best Practices

This IBM Redpaper provides examples on the best practices for developing new sites or migrating existing ones. The objective is to document how the tools can be combined with the systematic development method to streamline a WebSphere Commerce project.

Development and Migration Tools

For more information:
ibm.com/redbooks