APPLYING USE CASES

A PRACTICAL GUIDE

GERI SCHNEIDER
JASON P. WINTERS

Foreword by Ivar Jacobson

Applying Use Cases

Second Edition

The Addison-Wesley Object Technology Series

Grady Booch, Ivar Jacobson, and James Rumbaugh, Series Editors

For more information, check out the series web site at www.awprofessional.com/otseries.

Ahmed/Umrysh, *Developing Enterprise Java Applications with J2EE™ and UML*

Arlow/Neustadt, *Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML*

Arlow/Neustadt, *UML and the Unified Process: Practical Object-Oriented Analysis and Design*

Armour/Miller, *Advanced Use Case Modeling: Software Systems*

Bellin/Simone, *The CRC Card Book*

Bergström/Råberg, *Adopting the Rational Unified Process: Success with the RUP*

Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*

Bittner/Spence, *Use Case Modeling*

Booch, *Object Solutions: Managing the Object-Oriented Project*

Booch, *Object-Oriented Analysis and Design with Applications, 2E*

Booch/Bryan, *Software Engineering with ADA, 3E*

Booch/Rumbaugh/Jacobson, *The Unified Modeling Language User Guide*

Box/Brown/Ewald/Sells, *Effective COM: 50 Ways to Improve Your COM and MTS-based Applications*

Carlson, *Modeling XML Applications with UML: Practical e-Business Applications*

Collins, *Designing Object-Oriented User Interfaces*

Conallen, *Building Web Applications with UML, 2E*

D'Souza/Wills, *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*

Douglass, *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*

Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*

Douglass, *Real Time UML, 3E: Advances in The UML for Real-Time Systems*

Eeles/Houston/Kozaczynski, *Building J2EE™ Applications with the Rational Unified Process*

Fontoura/Pree/Rumpe, *The UML Profile for Framework Architectures*

Fowler, *Analysis Patterns: Reusable Object Models*

Fowler et al., *Refactoring: Improving the Design of Existing Code*

Fowler, *UML Distilled, 3E: A Brief Guide to the Standard Object Modeling Language*

Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML*

Gomaa, *Designing Software Product Lines with UML*

Graham, *Object Oriented Methods, 3E: Principles and Practice*

Heinckiens, *Building Scalable Database Applications: Object-Oriented Design, Architectures, and Implementations*

Hofmeister/Nord/Dilip, *Applied Software Architecture*

Jacobson/Booch/Rumbaugh, *The Unified Software Development Process*

Jordan, *C++ Object Databases: Programming with the ODMG Standard*

Kleppe/Warmer/Bast, *MDA Explained: The Model Driven Architecture™: Practice and Promise*

Kroll/Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*

Kruchten, *The Rational Unified Process, 3E: An Introduction*

Lau, *The Art of Objects: Object Oriented Design and Architecture*

Leffingwell/Widrig, *Managing Software Requirements, 2E: A Use Case Approach*

Manassis, *Practical Software Engineering: Analysis and Design for the .NET Platform*

Marshall, *Enterprise Modeling with UML: Designing Successful Software through Business Analysis*

McGregor/Sykes, *A Practical Guide to Testing Object-Oriented Software*

Mellor/Balcer, *Executable UML: A Foundation for Model-Driven Architecture*

Mellor et al., *MDA Distilled: Principles of Model-Driven Architecture*

Naiburg/Maksimchuk, *UML for Database Design*

Oestereich, *Developing Software with UML, 2E: Object-Oriented Analysis and Design in Practice*

Page-Jones, *Fundamentals of Object-Oriented Design in UML*

Pohl, *Object-Oriented Programming Using C++, 2E*

Pollice et al. *Software Development for Small Teams: A RUP-Centric Approach*

Quatrani, *Visual Modeling with Rational Rose 2002 and UML*

Rector/Sells, *ATL Internals*

Reed, *Developing Applications with Visual Basic and UML*

Rosenberg/Scott, *Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example*

Rosenberg/Scott, *Use Case Driven Object Modeling with UML: A Practical Approach*

Royce, *Software Project Management: A Unified Framework*

Rumbaugh/Jacobson/Booch, *The Unified Modeling Language Reference Manual*

Schneider/Winters, *Applying Use Cases, 2E: A Practical Guide*

Smith/Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*

Stevens/Pooley, *Using UML, Updated Edition: Software Engineering with Objects and Components*

Unhelkar, *Process Quality Assurance for UML-Based Projects*

van Harmelen, *Object Modeling and User Interface Design: Designing Interactive Systems*

Wake, *Refactoring Workbook*

Warmer/Kleppe, *The Object Constraint Language, 2E: Getting Your Models Ready for MDA*

White, *Software Configuration Management Strategies and Rational ClearCase™: A Practical Introduction*

The Component Software Series

Clemens Szyperski, Series Editor

For more information, check out the series web site at www.awprofessional.com/csseries.

Allen, *Realizing eBusiness with Components*

Apperly et al., *Service- and Component-based Development: Using the Select Perspective™ and UML*

Atkinson et al., *Component-Based Product Line Engineering with UML*

Cheesman/Daniels, *UML Components: A Simple Process for Specifying Component-Based Software*

Szyperski, *Component Software, 2E: Beyond Object-Oriented Programming*

Whitehead, *Component-Based Development: Principles and Planning for Business Systems*

Applying Use Cases

Second Edition

A Practical Guide

Geri Schneider
Jason P. Winters

ADDISON-WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data
Schneider, Geri.
Applying use cases: a practical guide / Geri Schneider, Jason P. Winters–2nd ed.
p. cm–(The Addison-Wesley object technology series)
Includes bibliographical references and index.
ISBN 0-201-70853-1
1. Application software—Development. 2. Use cases (Systems engineering) I. Winters, Jason P. II. Title. III. Series.

QA76.76.A65 S34 2001
005.1—dc21

00-052165

Text printed on recycled and acid-free paper.

ISBN 0201708531

9 1011121314 DOC 08 07 06

9th Printing February 2006

Contents

This page intentionally left blank

Foreword

When I first proposed a new set of modeling concepts back in 1967 as the result of my work on large telecommunication switching systems and system design, the idea of use cases as a method of analysis was very sketchy. With the emergence of object-oriented ideas and my subsequent work in applying OO in the 1980s and formalizing the principles underlying Objectory, use case analysis began to take better shape and to play a significant role in the analysis of the problem domain. Today the ideas embodied in use cases have matured, and this technique has become a significant tool that belongs in every analyst's toolkit.

With the incorporation of use cases into the industry standard modeling language, UML, it is time for a new book that illustrates the current notation and semantics of use cases in a practical, easy-to-understand manner. Use case analysis also plays a central role in the new Unified Process for software development. It is, therefore, critical that managers, architects, designers, analysts, domain experts, programmers, and testers understand how to apply use cases.

In *Applying Use Cases,* Geri Schneider and Jason Winters have done an excellent job of introducing this powerful technique and demonstrating its application in real-world settings. Rather than making everything perfect up front, the examples progress in much the same manner you would find in a real project, with early rough models being refined as the team gains understanding of the project. This realism allows the introduction of issues that would arise in actual projects. *Applying Use Cases* is easy to read, but contains a wealth of detail.

This book clearly reflects Geri's experience as a trainer for Rational Software, the time she has spent mentoring and training customers of Wyyzzk Training and Consulting, and the time Jason has spent using the techniques and mentoring engineers at Lucent Technologies. It is an excellent resource for anyone who needs to understand use case analysis, and I recommend it highly.

Ivar Jacobson

Preface to the Second Edition

There have been many changes for us and for the UML since the first edition was released in September 1998. The book has changed to stay current.

The material in the first edition is also in the second edition, but you may find it in a new location. We moved the engineering-oriented material to the end of the book, and the business-oriented material to the beginning. This should make it easier for different audiences to find the material that interests them.

We updated the book to UML 1.3. A lot of the changes are in Chapters 3 and 4 because that is where we described most of the notation. The uses relationship became two relationships in UML 1.3, include and generalization. The extends relationship became extend. In both cases the notation changed as well. The definition of scenarios changed a bit too. What we used to call scenarios are now called paths.

We have added some new material that we found useful and important. Chapter 6 is a new chapter on setting the level of detail in use cases. This includes information on business process-level use cases and maintaining traceability between use cases at different levels of detail. Chapter 7, Documenting Use Cases, includes some ideas on handling login and CRUD (create, read, update, delete) in use cases. Chapter 8, Reviews, has a new section on common mistakes we have seen and how to fix them. We have included more information on sequence diagrams in Chapters 5 and 9.

There have been changes for me and Jason as well. Jason left Octel and is now a staff engineer at Cadence Design Systems. I liked having my own business, but didn't like the bookkeeping, so I took a job running the OO division of Andrews Technology, Inc. We still have Wyyzzk and Jason does some weekend consulting for that business. Things even changed on the publishing

side. Addison-Wesley is now part of Pearson Education, and we have a whole new team managing the Object Technology series. They have been wonderful to work with and made the transition as smooth as possible.

One question we get asked a lot is: What do the footprints and people talking icons mean? The footprints mark major steps in the process. The people talking appear next to the storyline.

Thank you for all the e-mail about the book. We don't always get a chance to reply, but we have read all your letters and hope we have answered most of your questions in this second edition.

Many thanks to our distinguished reviewers. They worked as hard as we did to make this book happen:

- Lauren Thayer
- Venkat Narayanan
- Guy Rish
- Kelli A. Houston, Rational Software Corporation
- John Sunda Hsia

Speaking of hard workers, we were most fortunate to be working with Paul Becker, Ross Venables, Tyrrell Albaugh, and her production team at Addison-Wesley. Our most heartfelt thanks for all your support and patience. You guys did all the tough work to make this book a reality.

Finally, a few very special people. Thanks to Lauren Thayer and Kristy Hughes for being my constant friends for almost 15 years. And my cats, Patches and Joker, for keeping me company all those hours on the computer. And as always, thank you to Jason Winters for his love, support, and encouragement. You all are the wind beneath my wings.

Geri Schneider Winters
Santa Clara, California

Preface

You're about to start a new project. Sometimes it seems like colonizing the moon would be easier. But you assemble a stalwart team and prepare to set sail on the good ship *Requirements,* hoping to reach the fabled land of Success. They say there are no failed projects in Success, and the profit margin is so high, the streets are paved with gold.

There are many dangers between here and Success. Many a ship is sunk on the way—some say as many as 80 percent never reach that fabled land. You query those who have tried before. "Use a ship from the OO line," they say. "Booch, OMT, OOSE, UML are all good models to choose from. You'll also need a chart showing risks along the way and an architecture of the major land masses. And finally you'll need to plot a course of use cases to reach your destination."

Use Cases are included in the Unified Modeling Language and are used throughout the Rational Unified Process. They are gaining wide acceptance in many different businesses and industries. Most often, use cases are applied to software projects and enterprise-wide applications.

This book is for anyone interested in applying use cases to project development. While we can't guarantee you will always have successful projects when using use cases, we can give you another way of looking at the projects you are developing and some tools that will make success more likely.

You will get more benefit out of the book if you have some basic knowledge of object-oriented concepts. We will use the Unified Modeling Language for the notation, explaining the notation as we use it. A good book to use for reference on the notation is *UML Distilled* by Fowler. This is an excellent book on the topic and easy to read.

This book is organized using the Rational Unified Process as a framework. Within the phases of the process, we talk about the activities in the phase, focusing on activities based on use cases. We touch lightly on activities that interact with use cases, such as software architecture, project management, and object-oriented analysis and design. These are very important activities, with whole books devoted to each topic. Therefore, in the resource list in Appendix A, you will find our favorite books on these topics.

We have used one example, an order-processing system for a mail order company, throughout the book. This allows us to maintain consistency and build up a reasonably complex example. Parts of the solution are given in the various chapters to illustrate the concepts.

This book is presented as a sequence of steps, though life is never that simple. Each part will contribute to the rest until the system is complete. So if a section says to create an architecture, do what you can at that time, using what you currently know. You will add to it and refine it based on knowledge gained while working through the process.

You don't have to read the whole book before starting with use cases. Chapters 1 through 6 give the basics of working with use cases. We recommend that everyone reads those chapters. Chapter 9 covers architecture and mapping use cases into the architecture. Chapter 7 covers documenting use cases. Chapter 10 covers project planning with use cases, and Chapter 8 covers reviewing the use case documents. Chapter 11 goes into moving from use cases to OOAD.

Ultimately, use cases are about documenting your system. Plan on doing a lot of writing. Appendix A provides a list of books we reference throughout the text, as well as other books we have found useful when developing projects. Appendix B shows the document templates used. These provide an example and a starting point for your own project. Modify them as needed to work with your project.

△ ▽ △ ▽ △

In October of 1995, Rational Software Corporation merged with Objective Systems. Among other things, this merger brought with it Ivar Jacobson and his use cases. In February 1996, I wrote and delivered the first use case course for Rational, which combined use cases with the object-oriented methodologies of Grady Booch and Jim Rumbaugh. Since that time, I have taught and run workshops on use cases with many of Rational's customers, as well as customers of my consulting company, Wyyzzk Training and Consulting. As I have taught them, so they have taught me. This book came out of what I've learned through the workshops.

Chapter 1

Getting Started

Use cases are used to describe the outwardly visible requirements of a system. They are used in the requirements analysis phase of a project and contribute to test plans and user guides. They are used to create and validate a proposed design and to ensure it meets all requirements. Use cases also are used when creating a project schedule, helping to plan what goes into each release.

This book will give practical guidelines for applying use cases to a project. We will cover a project from its initial inception ("Hey! How about. . . .") to just before we actually start to build a system. We also will look at applying use cases in testing the system code and creating user manuals.

In this book we'll look at use cases from many viewpoints, showing how they contribute to the architecture, scheduling, requirements, testing, and documentation of a project. We'll look at the system from the user's point of view, discuss issues such as boundaries, interfaces, and scoping, and look at how to break a really large system into manageable chunks. We also will look at who would be interested in the documentation you'll be writing and what to look for in a review. We need to consider things such as how to build flexibility into a system, how to make a build-versus-buy decision, and how to turn the documents into an object-oriented design.

This book does not contain in-depth details about software architecture, project planning, testing, process, or methodology. Instead, you will find a listing of books we like on these topics in Resources (Appendix A). There are a number of good books on these topics; the resource list gives you just a starting point.

AN ITERATIVE SOFTWARE PROCESS

Use cases can be used in many processes. Our favorite is a process that is iterative and risk driven. It works well with use cases and object-oriented methodologies. It helps identify and address risks early in the process, leading to more robust and better quality systems. One commonly used iterative and risk driven process is the Rational Unified Process (RUP). We will give a very brief description of RUP here, showing where use cases fit into the process. Subsequent chapters will go into more detail on how use cases are used at each phase.

RUP is divided into four primary phases: inception, elaboration, construction, and transition.

During the inception phase you will determine the scope of the project and create a business case for it. At the end of the inception phase you should be able to answer the question, Does it make good business sense for us to continue with this project?

During the elaboration phase you will do requirements analysis and risk analysis, develop a baseline architecture, and create a plan for the construction phase.

During the construction phase you will progress through a series of iterations. Each iteration will include analysis, design, implementation, and testing.

During the transition phase you will complete the things that make what you developed into a product. These can include beta testing, performance tuning, and creating additional documentation such as training, user guides, and sales kits. You will create a plan for rolling out the product to the user community, whether internal or external.

So where do use cases fit into all this? In the inception phase, high-level use cases are developed to help scope out the project: What should be included in this project, and what belongs to another project? What can you realistically accomplish given your schedule and budget?

In the elaboration phase, you will develop more detailed use cases. These will contribute to the risk analysis and the baseline architecture. The use cases will be used to create the plan for the construction phase.

In the construction phase, you will use use cases as a starting point for design and for developing test plans. More detailed use cases may be developed as part of the analysis of each iteration. Use cases provide some of the requirements that have to be satisfied for each iteration.

In the transition phase, you will use use cases to develop user guides and training.

AN EXAMPLE PROJECT

Throughout this book we will use an example project. We will work through all the techniques using the same example. The notation we will use is the Unified Modeling Language (UML), which is outlined in Appendix C.

The example we will be following is for an order-processing system for a mail order company. Let's start at the very beginning, when four friends gather around a table after dinner and someone gets an idea.

"This is crazy!" Dennis exclaimed, sitting down next to Tara, almost spilling his coffee.

"What is?" Lisa asked, sitting down with Gus and her own cup of mocha.

"The fact that I can't find a single supplier that will give me reasonable service and parts without all the headaches! I've got one supplier who has great service, and I like dealing with him. But he takes three weeks to get me even the simplest order! And the other one—oh, they're something else. Sure, I can get orders within three days, but half the time the orders are wrong, and when I call them back about it, they make it sound like it's my fault! It's almost as if they are *trying* to make me go elsewhere."

"Yes, I know what you mean," Lisa said. "I've had my own problems with mail order companies. You'd think they would pay more attention to their customers!"

"I really think I could do a better job myself. I sure know a lot about what not to do."

This had been a common complaint from Dennis in the last several months, and by now the group was well acquainted with it. Tara suddenly smiled and piped up with "Why don't you start one?" "Start one what?" Dennis muttered into in his coffee.

"Start a mail order company! What would you do to fix the problems you've seen?"

"Well, it seems like automating the order processing would help a lot. It would also let me run the company myself for a while. But I don't know anything about software."

At this point, Gus joined the conversation. "You need to plan it out. I learned a method in my OO class we could use. And we could help! By putting us and all of our experiences together, we could figure out how to use our different skills in the right places and work out the sections we don't know!"

"OO? What's that? You know I'm not a programmer. I don't know anything about programming languages."

"No, OO isn't about a programming language. It's a way of thinking about a problem, a way of modeling and breaking it down into identifiable objects so you can work with them. It really doesn't matter what the problem is, whether it's a programming problem or a problem like starting a new business. It's just an approach on how you look at it."

"Hmmm. . . ." mused Dennis, liking the idea the more he thought about it. "And you would all be willing to help?"

"Sure!"

"Why not?"

"Sounds like fun!"

"Well . . . okay! So, Gus, where do we start with this all-dancing-all-singing magical OO process?"

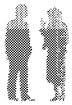
Before you can write use cases, you have to gather some information that will provide a starting point. This is part of the inception phase of the RUP. The information you collect includes a project description, market factors that affect your project, risk factors for your project, and assumptions you are making. The rest of this chapter will touch on all these sources of information.

THE PROJECT DESCRIPTION

So you have an idea for a project. The next step is to write out a description of what you plan to do. It sounds simple, and it can be. But the larger the group you have writing this description, the longer it will take and the more complex it will be. It is best to have just a couple of people write out a brief, but complete, description of the project.

The project description should range in size from one short paragraph for a small project up to no more than a couple of pages for a really large project. This is not a description of the requirements, but a description of the project in general.

The biggest mistake made at this point is not writing the description. Usually this is because everybody thinks they know what the project is about, so why write it down? We have spent several days in meetings while the project team has argued about what a one-paragraph description should say. Until you write it out, you can't be sure everyone agrees on the same project description.



"So, let's get started. We need to write out a description of order processing in normal everyday language. This will become the problem statement, a way to start getting the requirements for the project."

"Why do we need to write it out? We're all familiar with ordering products from mail order companies. This seems too formal for such a simple problem."

"Well, we should write it all down to make sure everyone has the same idea. Even if you're working alone, it's still a good idea to write it down so you don't leave out something important. Besides, it'll give us someplace to start, and we can add to it as we go along. Here, I'll start."

Problem Description

> We are developing order-processing software for a mail order company called National Widgets, which is a reseller of products purchased from various suppliers.
>
> Twice a year the company publishes a catalog of products, which is mailed to customers and other interested people.

"You think twice a year is good? What if our products change faster than that?"

"Remember, this is just to start us off. We will add to it and change it as we get farther along and understand more about what's going on. Let's keep going."

More Requirements

> Customers purchase products by submitting a list of products with payment to National Widgets. National Widgets fills the order and ships the products to the customer's address.
>
> The order-processing software tracks the order from the time it is received until the product is shipped.
>
> National Widgets provides quick service. They should be able to ship a customer's order by the fastest, most efficient means possible.

"Great! That looks like us! Now, what else should we do?"

What did our friends do right? They kept the description brief, talking about what they want to accomplish, not how to do it. They wrote down the elements important to them: It's a catalog company—a reseller, not a manufacturer; it provides quick service; and the software is used throughout the process to track orders. These are the key characteristics of their project. They haven't worried about making their description perfect. If there were something that they should NOT do, they would have written that down as well. They now have a basic description that they agree on but that may need to be modified later. However, the key characteristics should not change.

STARTING RISK ANALYSIS

Now that you have a description, the next step is to write down other things you know about your project. You are looking for marketing factors that will influence your project, good or bad, and anything that could cause the project to fail or to be rejected by the customer. We will use these with the problem statement to create use cases, other requirements, and risk factors. Start by considering market factors:

- Who or what the competition is
- What technologies you are depending on, such as:
  - Web
  - Object databases
  - Power PC chip
- Market trends that influence your project
- Future trends you are depending on, such as:
  - More home offices
  - More small companies
- Is it possible to be:
  - Not fast enough to market
  - Too fast to market

For our mail order company we can create a list of market factors based on personal experience.

Mail Order Market Factors

In most households, all adults work at least part-time. They have less time available for shopping, so they usually are willing to pay for conveniences such as purchase delivery.

Web shopping and home-shopping networks are popular and are competitors in this market.

Other mail order companies provide 24-hour order takers, delivery times ranging from overnight to two weeks, gift wrap, and volume discounts.

Be creative as you make this list. Brainstorm a lot. Put down just about anything you can think of. Put down the wildly unlikely as well as things that will probably happen. The idea at this stage is to look at the project from many viewpoints. This will help solidify your ideas. Look at some books on marketing trends for ideas. What are competing companies doing right or doing wrong?

You also need to consider risk factors in your project. You need to include things that can go wrong. Writing down only the things that you'd *like* to happen is a sure recipe for disaster. It is far better to think of how things can go wrong so you can plan for them than to wander along and be surprised.

You also need to include the possibility of being wildly successful. Sometimes you'll find that things actually can go wrong if you are too successful. For example, what would happen to National Widgets if, in the first month, they get 4,000 calls? They will now have to handle multiple order takers and large amounts of data. Presuming the company can handle this surge, can the software handle it? The company could lose business if the software is not up to the demands placed on it, possibly getting a bad reputation because of it. Our friends will write this down as one of their risks.

Here are some things to consider as possible risk factors:

- People
  - Team not experienced
  - Team not familiar with the technologies to be used
  - Unable to hire people with the right background
- System
  - Number of transactions per time frame
  - Number of expected users
  - Expected duration of some functionality
  - Legacy systems you have to interface with, such as:
    - Software
    - Business processes
    - Data stores, databases
- Resource
  - Too short a schedule
  - Too many users
  - Supplier can't or won't deliver product we depend on
- Technology
  - Dependence on a technology that changes
- Corporate
  - Lack of user acceptance
  - Too fast company growth

Our risk factors for the order-processing project take into consideration the inexperience of the team, system failure, and the needs of the market.

National Widgets Risk Factors

- Some of the people designing the software are inexperienced.
- How can we prevent lost orders on system failure?
- The system has to be easy for nontechnical people to use.
- Can we be successful if we don't support a Web interface?
- What if the system is immediately flooded with orders?
- How do we handle many simultaneous users in different parts of the company?
- How do we handle the database crashing?

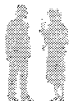
Take this list of risks and the list of market factors, eliminate extremes such as a comet crashing into your company and putting you out of business or the sun failing to rise, and prioritize the rest. The new prioritized list you've created is your first risk analysis. All the things you've listed on it put you at risk for not completing your project. The more serious items are listed first, with less severe risks at the bottom. The high-risk factors must be addressed if you expect your project to succeed.

Looking at National Widgets risk factors, we want to mark the items high risk if we are fairly sure we will fail unless those risks are addressed, and we are fairly certain that these risks will happen. We chose three risks as high risk for this project: lack of team experience, ease of use for nontechnical people, and support for a Web interface. We picked one medium risk: the need for multiple users to access the system at the same time. This is a real need, but there are many good technical solutions to the problem, some of which we could just purchase and incorporate into our system. This is less likely to cause project failure than lack of team experience. The other risk factors we marked low. The system flooded with orders is judged to be highly unlikely for this company, and the database or system crashing should be fairly unlikely as well. Experience suggests that if an order gets lost, the customer is likely to call us looking for it, so we can recover data that way.

You may pick the priorities differently. Each company and project will have different ways of looking at the problem, different factors that are important. Also, remember that this is just a starting point and that you'll be adding to it as you continue.

As part of your risk list, put together and maintain a list of assumptions. These are decisions you make with little or no hard data. You may need to pick one way of doing something based on gut feel or experience. Record that decision and why you made it. This list should be reviewed regularly. Some things will remain as assumptions. For others, you will be able to get hard data on which to base your decisions. Remove things that are no longer assumptions and add the new assumptions you've made.

Let's go back and see how the group members are doing with their lists.



"Okay, it's been a busy evening. Let's see what we've gotten out of it." Gus handed around the lists shown in Exhibit 1-1.

"Wow," said Dennis. "We sure can fail in a lot of ways. But how has this helped us define the software? So far, all it's done is make me worry that we forgot something."

"Don't worry, Dennis. We're just getting started. We want to find things that could make us fail now so we can fix the problems rather than being surprised by them later."

Exhibit 1-1 Order-Processing Problem Statement

Problem Description

- We are developing order-processing software for a mail order company called National Widgets, which is a reseller of products purchased from various suppliers.
- Twice a year the company publishes a catalog of products, which is mailed to customers and other interested people.
- Customers purchase products by submitting a list of products with payment to National Widgets. National Widgets fills the order and ships the products to the customer's address.
- The order-processing software tracks the order from the time it is received until the product is shipped.
- National Widgets will provide quick service. They should be able to ship a customer's order by the fastest, most efficient means possible.
- Customers may return items for restocking but will sometimes pay a fee.

Assumptions

- An electronic interface, such as the Web, would be good for some customers.
- We expect to use multiple shipping companies and insured methods.

Risk Factors

- *High:*
- Some of the people designing the software are inexperienced.
- The system has to be easy for nontechnical people to use.
- Can we be successful if we don't support a Web interface?

- *Medium:*
- How do we handle many simultaneous users in different parts of the company?
- *Low:*
- How can we prevent lost orders on system failure?
- What if the system is immediately flooded with orders?
- How do we handle the database crashing?

Exhibit 1-1 Order-Processing Problem Statement (*Continued*)

Market Factors

- In most households, all adults work at least part-time. They have less time available for shopping, so are usually willing to pay for conveniences such as purchase delivery.
- Web shopping and home shopping networks are popular and are competitors in this market.
- Other mail order companies provide 24-hour order takers, delivery times ranging from overnight to two weeks, gift wrap, and volume discounts.

CHAPTER REVIEW

Table 1-1 shows deliverables you should have completed at this point. Your risk analysis should include known risks, other known market factors, and assumptions you have made about the project.

These are just preliminary versions, showing what you know right now. You will modify these things as you learn more about your project. The next chapter covers using use cases to find the boundaries of the system and the scope of the project.

Table 1-1 Inception Phase Deliverables

Complete	Deliverables
✔	Project description
✔	Risk analysis
	Use case diagram
	Description of actors and use cases
	Project proposal

Index