Developing Applications with Java™ and UML

By Paul R. Reed Jr.

Start Reading

Publisher: Addison Wesley
Pub Date: November 01, 2001
ISBN: 0-201-70252-5
Pages: 504
Slots: 1

- Table of Contents

Developing Applications with Java(TM) and UML focuses on the craft of creating quality Java software. The book introduces the fundamentals of the Unified Modeling Language (UML) and demonstrates how to use this standard object-oriented notation to build more robust Java applications that fulfill user requirements and withstand the test of time.

The book features the Rational Unified Process and uses a large-scale application to illustrate the development process, showing you how to establish a sound project plan, gather application requirements, create a successful Java design with UML, and implement Java code from UML class and sequence diagrams. This sample application showcases the latest Java technology frameworks, including Java Server Pages(TM) (JSP(TM)), servlets, and the Enterprise JavaBeans(TM) (EJB(TM)) 2.0 server-side technology.

With this book you will learn how to:

- Estimate projects with accuracy and confidence
- Map UML to Java-based deliverables
- Understand and describe application requirements using UML use cases
- Create a design based on UML class and sequence diagrams
- Use Rational Rose to create and track UML artifacts and generate skeletons for component code
- Build server-side Java functionality using JSP, servlets, and EJB 2.0 beans
- Produce code using several options, including JavaBeans, EJB Session Beans, and EJB Entity Beans (using both Bean-Managed Persistence and Container-Managed Persistence)
- Explore the benefits of deploying Java applications on both open-source and commercial application server

products

Based on the author's extensive professional experience and today's most advanced software development methods, Developing Applications with Java(TM) and UML teaches you how to use UML and the latest developments in technology to create truly successful, professional-quality Java applications.

Preface

This book focuses on the most powerful approach available today to modeling and building industrial-strength Java applications: the Unified Modeling Language (UML) adopted in 1997 by the Object Management Group (OMG). A project lifecycle and software process model are demonstrated (Rational's Unified Process) through a sample application from requirements gathering, using use-cases, through implementation via the creation of Java code from class and sequence diagrams. This sample application uses two different architectures that the reader can focus on, depending on their specific needs. The first architecture uses Java servlets, JavaServer Pages (JSP), and JavaBeans all running in the freely available Apache Tomcat server. The second uses Java servlets, JavaServer Pages (JSP), and Enterprise JavaBeans 2.0 (EJB) all running in a commercially available application server.

It took me many years to understand that writing a program is nothing more than a learned tactical skill. To program in a language like Java is to be a journeyman. But to capture someone's requirements in an intelligent fashion and organize the necessary resources and resulting software into a cohesive deliverable are the signs of a strategic craftsman.

To me, the majority of Java books never consider Java "in the large." They focus on the small view, covering single Java-enabled extensions such as JavaBeans, servlets, and JavaServer Pages. Although these views, too, are necessary, unfortunately no one seems to touch on project planning, software process, and the methodology for building enterprise-status Java applications. This is a difficult topic to explore and present because the whole subject of process spurs on many heartfelt debates and opinions. At the urging of many of my colleagues and supportive readers of my first book, *Developing Applications with Visual Basic and UML*, I have undertaken a similar project for Java.

Who Should Read This Book

This book is intended for anyone who wants to successfully build Java applications that can stand up over time. It provides an accurate road map for anyone to achieve the following goals:

- Review two processes: one commercially available through Rational Software called the Unified Process and one from my own experiences called Synergy. The greatest emphasis will be placed on the Unified Process.
- Establish a sound project plan (presented in depth in Appendix A).
- Estimate projects with confidence, rather than by using a rule-of-thumb approach.
- Understand and describe the requirements of the application using UML use-cases.
- Create a sound design based on UML class and sequence diagrams.
- Use a visual modeling tool such as Rose by Rational Software not only to create and track UML artifacts, but also to generate skeleton component code. Although I firmly believe that an automated code generation process is a big factor contributing to successful projects, it is certainly not mandatory.
- Use Java to build server-side Java functionality employing architectures such as JavaServer Pages (JSP) and servlets, along with either JavaBeans or Enterprise JavaBeans 2.0 (EJB) managing the business rules.
- Produce the code for the project using an evolutionary approach showing various technology options: (1) servlets, JSP, and JavaBeans; (2) servlets, JSP, and bean-managed persistence (BMP); and (3) servlets, JSP, and container-managed persistence (CMP).
- Investigate the benefit of deploying Java applications on both open-source products like the Apache Tomcat server and commercial application server products such as BEA's WebLogic application server.

Anyone building Java applications today needs this book.

What You Need to Know to Use This Book

Maybe it's best to start out with what you don't need to know to benefit from this book. First, you don't need to know anything about UML. I present the essential aspects of UML and, more importantly, how they relate to Java deliverables. Although UML is expressed through nine separate diagrams, you will benefit the most from a core set.

Second, you don't need a formal background in object-oriented concepts (but it certainly doesn't hurt). I discuss standard object constructs in Chapter 2.

Third, you should have some conversational understanding of what Enterprise JavaBeans is. For a really thorough treatment of Enterprise

JavaBeans (EJB), you should focus on one of the many texts that cover them in more detail. A favorite of mine is a book by Richard Monson-Haefel entitled *Enterprise JavaBeans*, published by O'Reilly. You will also benefit from some exposure to JavaServer Pages (JSP). One of my favorite sources on this topic is a book by Hans Bergsten entitled *Java Server Pages*, also published by O'Reilly.

This book assumes that you have a working knowledge of Java. Both the new Java programmer and the experienced Java programmer will benefit. I don't cover the basics of simple Java constructs, assuming that you already know these. I do briefly review the tenets of Java's support for object-oriented principles in Chapter 2, but only as a baseline for other topics related to UML. If you have had no exposure to Java, buy this book anyway and open it after you have had some initial training in that programming language.

This book emphasizes the most mainstream Java techniques and products that are used to build production applications. When I began this book, I planned to cover all kinds of Java technologies (i.e., applets, Java applications talking to servlets or JSPs). However, it quickly became apparent to me that the majority of my clients and my associates' clients were all pretty much cut from the same mold in terms of architecture. They consist of a light client browser on the front end (with minimal JavaScript for syntax editing), and a Web server intercepting those browser requests with either servlets and/or JavaServer Pages acting as a broker within a container product that houses the business rules. These business rules are implemented as either JavaBeans or Enterprise JavaBeans. The container products range from open-source solutions like Apache Tomcat to commercial products.

The two biggest of the commercial application server players I run across are BEA (with its WebLogic product) and IBM (with its WebSphere product). This doesn't mean there aren't other good commercial container products, but these two vendors have the lion's share of the market. This book will utilize a light client-side technology (no applets or Java applications), and a Web server running servlets and JavaServer Pages, which in turn send messages to either JavaBeans (Tomcat) or Enterprise JavaBeans (session and entity beans) residing in a commercial application server.

In the case of the latter, I have chosen to use BEA's WebLogic as my application server. Don't be discouraged if you are using another vendor's application server product because this book's coverage of EJB is based on the 2.0 specification. This release of EJB resolved many of the ambiguities that kept beans from being truly transportable across vendor

implementations. So regardless of your EJB vendor, you will be able to use the code built in this book.

It would be unfair to say that you will know everything about EJBs after reading this book. If you already know about EJBs, this book will help you put them into a sound design architecture. The emphasis is on the notation (UML) and the process (Unified Process and Synergy) in beginning, developing, and implementing a software project using the Java language. The benefit of seeing an application from requirements gathering to implementation is the key goal of this book. This is where I shall place my emphasis.

Structure of the Book

The following sections summarize the contents of each chapter.

Chapter 1: The Project Dilemma

Chapter 1 reviews the current state of software development and my reasoning regarding why it's in the shape that it is today. It also reviews the concept of iterative and incremental software development and provides an overview of both Rational Software's Unified Process and my Synergy Process methodology. In addition, it touches on the primary components of UML that will be covered in more depth later in the book.

Chapter 2: Java, Object-Oriented Analysis and Design, and UML

Chapter 2 covers some of the benefits of adopting Java as a development environment, presented in the context of Java's implementation of encapsulation, inheritance, and polymorphism. It then maps UML to various Java deliverables. Highlights include mapping the UML diagrams to Java classes and Java interfaces; mapping use-case pathways to Java classes; and mapping component diagrams to Java classes and Java packages.

Chapter 3: Starting the Project

Chapter 3 explores the case study used in the book: Remulak Productions. This fictional company sells musical equipment and needs a new order entry system. The chapter introduces a project charter, along with a tool, called the event table, to help quickly solidify the application's features. Further, the chapter maps events to the first UML model, the use-case.

Chapter 4: Use-Cases

Chapter 4 reviews the use-case, one of the central UML diagrams. Included is a template to document the use-case. Actors and their roles in the use-cases are defined. The concept of use-case pathways, as well as the project's preliminary implementation architecture, is reviewed. Also reviewed is an approach to estimating projects that are built with the use-case approach.

Chapter 5: Classes

Chapter 5 explores the class diagram, the king of UML diagrams. It offers tips on identifying good class selections and defines the various types of associations. It also covers business rule categorization and how these rules can be translated into both operations and attributes of the class. Finally, it discusses the utilization of a visual modeling tool as a means to better manage all UML artifacts.

Chapter 6: Building a User Interface Prototype

Chapter 6 reviews unique user interface requirements of each use-case. It develops an early user interface prototype flow and an eventual graphical prototype. Finally, it maps what was learned during the prototype to the UML artifacts.

Chapter 7: Dynamic Elements of the Application

Chapter 7 discusses the dynamic models supported by UML, exploring in depth the two key diagrams, often referred to as the interaction diagrams: sequence and collaboration. These are then directly tied back to the pathways found in the use-cases. Other dynamic diagrams discussed include the state and activity diagrams.

Chapter 8: The Technology Landscape

Chapter 8 covers the importance of separating logical services that are compliant with a model that separates services. It explores technology solutions specific to the Remulak Productions case study, including distributed solutions and the Internet using HTML forms, JSP, and servlets. Both JavaBeans and Enterprise JavaBeans as solutions for housing the business rules are also explored.

Chapter 9: Data Persistence: Storing the Objects

Chapter 9 explores the steps necessary to translate the class diagram into a relational design to be supported by both Microsoft SQL Server and Oracle databases. It offers rules of thumb regarding how to handle class inheritance and the resulting possible design alternatives for translation to an RDBMS. This book will deliver solutions that range from roll-your-own persistence using JavaBeans and JDBC, all the way to container-managed persistence (CMP) features of the EJB 2.0 specification. The latter removes all the requirements of the application to write SQL or control transactions. This chapter introduces the concept of value objects to reduce network traffic, as well as data access objects that encapsulate SQL calls.

Chapter 10: Infrastructure and Architecture Review

Chapter 10 finalizes the design necessary to implement the various layers of the application. It also presents the communication mechanism utilized between the layers and possible alternatives. Each class is delegated to one of three types: entity, boundary, or control. These types are used as the basis for the design implementation and as the solution to providing alternative deployment strategies.

Chapter 11: Constructing a Solution: Servlets, JSP, and JavaBeans

Chapter 11 builds the first architectural prototype for Remulak and does not rely on Enterprise JavaBeans. With the *Maintain Relationships* use-case as the base, the various components are constructed. The primary goal of the architectural prototype is to reduce risk early by eliminating any unknowns with the architecture. This chapter uses the Apache Tomcat server and introduces the concepts of user interface and use-case control classes.

Chapter 12: Constructing a Solution: Servlets, JSP, and Enterprise JavaBeans

Chapter 12 initially uses Rational Rose to generate EJB components. A primer on EJB is offered, along with a thorough discussion of the transaction management options in the EJB environment. Session beans are utilized as the use-case controller. Solutions that incorporate both container-managed persistence (CMP) and bean-managed persistence (BMP)

are presented. Leveraging the data access objects created in Chapter 11 is crucial to the success of a BMP implementation.

Updates and Information

I have the good fortune to work with top companies and organizations not only in the United States, but also in Europe, Asia, and South America. In my many travels, I am always coming across inventive ideas for how to use and apply UML to build more-resilient applications that use not only Java, but also C++, C#, and Visual Basic. Please visit my Web site, at www.jacksonreed.com, for the latest on the training and consulting services that I offer, as well as all of the source code presented in this book. I welcome your input and encourage you to contact me at prreed@jacksonreed.com.

About the Author

Paul R. Reed, Jr., is president of Jackson-Reed, Inc., www.jacksonreed.com, and has consulted on the implementation of several object-oriented distributed systems. He is the author of the book *Developing Applications with Visual Basic and UML*, published by Addison-Wesley in 2000. He has published articles in *Database Programming & Design*, *DBMS*, and *Visual Basic Programmer's Journal*, and he has spoken at industry events such as DB/Expo, UML World, and VBITS. He is also the author of many high-technology seminars on such topics as object-oriented analysis and design using UML, use-case analysis, the Unified Process, Internet application development, and client/server technology. He has lectured and consulted extensively on implementing these technologies to companies and governments in the Middle East, South America, Europe, and the Pacific Rim.

Paul holds an MBA in finance from Seattle University. He also holds the Chartered Life Underwriter (CLU) and Chartered Financial Consultant (ChFC) designations from the American College and is a Fellow of the Life Management Institute (FLMI).

Acknowledgments

All of us today can trace our perceived success through the company we keep. I owe my good fortune to the many relationships, both professional and personal, that I have developed over the years.

I would like to thank Coby Sparks, Dale McElroy, Richard Dagit, Mike "mikey" Richardson, John "the japper" Peiffer, Kurt Herman, Steve "glacier man" Symonds, Jeff Kluth (a.k.a. Jeffery Homes of this book, backpacking buddy and the owner of the real Remulak, a DB2 consulting firm), Dave "daver" Remy, David Neustadt, John Girt, Robert Folie, Terry Lelievre, Daryl Kulak, Steve Jackson, Claudia Jackson, Bryan Foertsch, Debbie Foertsch, and Larry Deniston (the OO guru from Kalamazoo) and the other folks at Manatron (www.manatron.com). Thanks to Ellen Gottesdiener of EBG Consulting (www.ebgconsulting.com) for her wonderful assistance on project charters, use-cases, and business rules. Thanks to Marilyn Stanton of Illuminated Consulting (www.illuminatedconsulting.com) for her continued support and advice. A big thank-you to some very good friends: Bill and Cindy Kuffner, Nick and Peggy Ouellet, Bev Kluth, Bill and Sayuri Reed, Betsy Reed, Rodney Yee (my wine mentor), and Brian and Susan Maecker (besides being one of the top realtors in the country at www.maecker.com, Brian shares my passionate pursuit of the perfect glass of red wine).

Other individuals warrant commendation for shaping my thoughts about sound Java design and the use of its various technology offerings. In particular, I would like to thank Hans Bergsten for his excellent advice on the topic of JavaServer Pages, and Floyd Marinescu for the creation of his fantastic Web site (www.theserverside.com). Floyd's many articles on design patterns applicable to applications using Enterprise JavaBeans (EJB) were very helpful to me. I would like to thank Eric Stahl and Tyler Jewell, both of BEA Systems, for their assistance in using their marvelous products. I want to acknowledge Per Kroll from Rational Software for his assistance with using Rational's tools.

I have the benefit of working with some great clients in both the United States and abroad. I would like to explicitly thank Vicki Cerda at Florida Power Light; Sara Garrison at Visa USA; all my friends at the Federal Reserve Bank of San Francisco, including Mike Stan, Linda Jimerson, Matt Rodriguez, and Bill Matchette; and finally, my good friend and Asian partner Lawrence Lim at LK Solutions (Singapore, Malaysia, and United Arab Emirates, www.lk-solutions.com).

Finally, special thanks to Paul Becker of Addison-Wesley for putting up with my "whining" during this project. I would also like to thank Diane Freed of Diane Freed Publishing Services for helping organize the production of this book. I want to give a special thanks to Stephanie Hiebert for her wonderful copyediting and for providing clarity to my most jumbled collection of prose.

The real heroes of this project are the reviewers 樫 oe Chavez, Hang Lau, Prakash Malani, Jason Monberg, Nitya Narasimham, and David Rine 栲 ho slugged their way through my early drafts and weren't shy at all to cast darts when necessary.

Chapter 1. The Project Dilemma

IN THIS CHAPTER

I have run across more projects after the fact that missed their initial goals than met them. One reason is that most of the project teams had no clue about what a development process was or how to customize one for a project's unique characteristics. In addition, most of the projects had little in the way of analysis and design artifacts to show how they got where they were. The whole endeavor lacked the ability to be tracked; that is, it lacked traceability.

This chapter lays the groundwork for the need of a software process. I will present two processes in this book: one that is commercially available from Rational Software called the Unified Process, the other based on my own experiences, which I call the Synergy Process. For reasons to be covered later, the Synergy Process will be presented in Appendix B. The primary process that will guide this book's efforts is the Unified Process, which is presented in greater depth in Appendix A.

This process, along with the Unified Modeling Language (UML), can ensure that your next Java projects have all of the muscle they need to succeed. More importantly, these projects will stand the test of time. They will be able to flex and bend with shifts in both the underlying businesses they support and the technology framework upon which they were built. They won't be declared legacy applications before they reach production status.

GOALS

- To review the dilemma that projects face.
- To explore the nature of an iterative, incremental, risk-based software development process.
- To become acquainted with the software process model used in this book, called the Unified Process.
- To examine how the project team can market the use of a process to project sponsors.
- To review the Unified Modeling Language and its artifacts, and how it serves as the primary modeling tool for a project's process.

The Sad Truth

The premise of my first book, *Developing Applications with Visual Basic and UML,* was that most software projects undertaken today don't come close to meeting their original goals or their estimated completion dates. My reasoning was that most project teams have a somewhat cavalier attitude toward project planning and software process. In addition, most projects have little in the way of analysis and design artifacts to show how they got where they are. That is, projects traditionally lack traceability. This holds true for applications built in any language梩ava included.

My professional career with computers began after college in 1979, when I began working on large IBM mainframe applications using technologies such as IMS and later DB2, what many people today would call legacy applications. However, I prefer the terms *heritage* or *senior* to *legacy.*

Not only did I get to work with some really great tools and super sharp people, but I also learned the value of planning a project and establishing a clear architecture and design of the target application. I saw this approach pay back in a big way by establishing a clear line of communication for the project team. But more importantly, it set in place the stepping-stones for completing a successful project.

In 1990 I worked on a first-generation client/server application using SmallTalk on the OS/2 platform. This was the start of a new career path for me, and I was shocked by the "process" used to build "production" applications in the client/server environment. The planning was sketchy, as was the delivery of analysis and design artifacts (something that showed why we built what we built).

This pattern of "shooting from the hip" software development continued with my use of PowerBuilder, Visual Basic, and later Java. The applications delivered with these products worked, but they were fragile. Today many applications wear the *client/server* or *distributed* moniker when they are just as much a legacy as their mainframe counterparts, if not more so. Even worse, many of these become legacy applications a month or two after they go into production. The fault isn't with the tool or the language, but with the lack of a sound process model and methodology to ensure that what is built is what the users actually want and that what is designed doesn't fall apart the first time it is changed.

Most organizations today ship their staff off to a one-week Java class and expect miracles on the first application. Take this message to heart: The fact that you know Java doesn't mean you will necessarily build sound object-oriented applications. If you don't have a sound process in place and a very firm footing in sound object-oriented design concepts, your application will become a Neanderthal waiting in line for extinction.

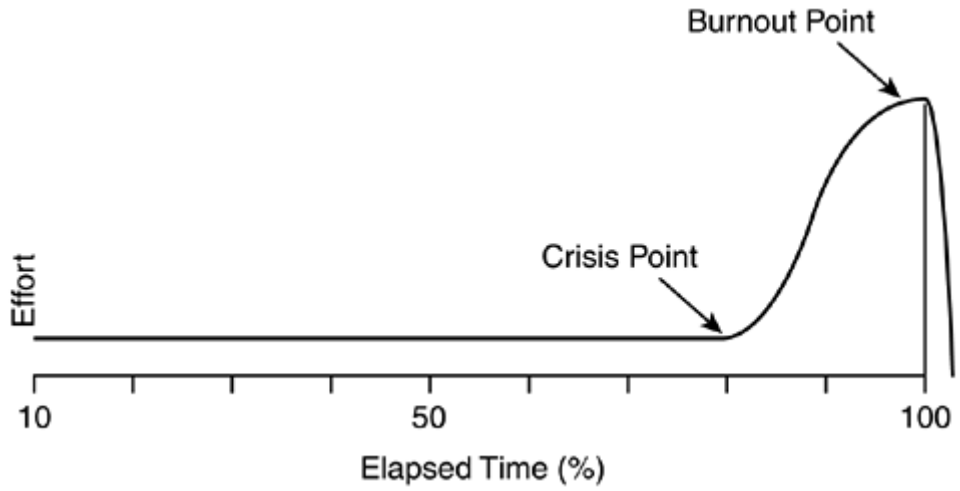
Slowly I began to apply my own opinions about process and methodology to the applications built in these environments. This worked quite well. The applications were more resilient and accepted change more easily, and the users typically had smiles on their faces.

This book combines all of my experience building distributed applications with UML, which I feel is the best artifact repository for documenting the analysis and design of an application today. I would also like to think that my approach to this topic is exciting because I use a real example throughout the book utilizing various Java technologies and tools to demonstrate how you might approach solving some of your own problems.

The Project Dilemma

Few projects proceed totally as planned. Most start out with much enthusiasm but often end in a mad rush to get something out the door. The deliverable is often hobbled with inaccuracies, or worse, it invokes unrepeatable responses from the project sponsors. Figure 1-1 shows a time line/effort comparison of person-hour expenditures for most projects that don't meet their commitments.

Figure 1-1. Time line of a typical project



You know the drill: The project plants a stake in the ground, with a big-bang deliverable two and half years away. And then you plod along, making adjustments and adding functionality until, toward the end, you realize, "We aren't going to make it." You are so far off the mark that you start adding more pounds of flesh to the effort. Before long, you begin to jettison functionality 梖or example, reporting, archiving, security, and auditing activities. You end up with a poor deliverable that adds to the black eye of the Information Technology (IT) department and further convinces the project sponsors that you can't develop software at all, let alone on time and within budget.

Unfortunately, like lemmings racing to the sea, companies repeat this saga again and again and again.

Iterative and Incremental Software Development

This dilemma of projects failing or not delivering promised functionality stems from the unwillingness of both the IT department and the project sponsors to take a learn-as-you-go approach to software development, more formally known as **iterative and incremental software development.**

In the context of a software project, many people confuse the terms *iterative* and *incremental.* The *American Heritage Dictionary,* Second College Edition, defines these terms as follows.

Iterative ?3….procedure to produce a desired result by replication of a series of operations that successively better approximates the desired result.

Incremental ?1. An increase in number, size, or extent. 2. Something added or gained….4. One of a series of regular additions or contributions.

Let's give these academic definitions a bit of a software flavor:

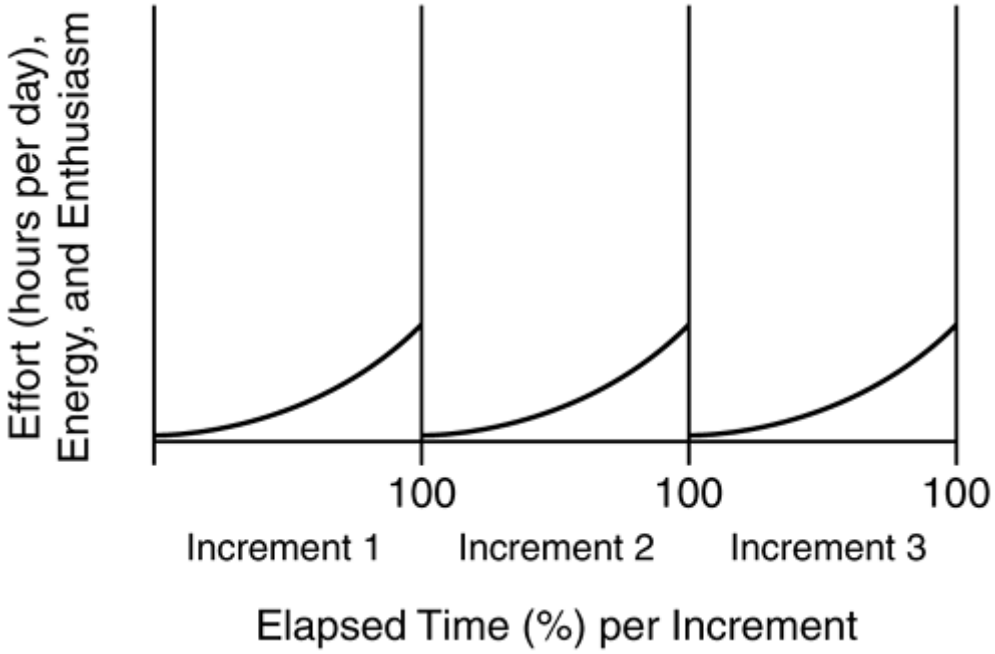
Iterative: The application of tasks in a repetitive fashion that works toward bettering an interim or final product.

Incremental: The creation of interim deliverables such that each one adds significant value to the overall project, stands on its own or operates within a well-defined interface, or might take part as a subcomponent of a final product.

As an example, suppose you are constructing a wooden play set for children. You begin by simplifying the project by breaking it up into two incremental parts: (1) tower and attached slide chute and (2) swing and trapeze frame. You realize the project by iterating through the building of each increment. The iterations might be first to create a detailed drawing; then to buy, measure, and cut the wood; next to bolt together the pieces; and finally to stain the wood. Each iteration improves the chances of producing a product that stands on its own. This approach is powerful because many of the same iterative tasks (drawing, sawing, bolting, and staining) are to be applied to each increment (tower/slide and swing/trapeze).

The challenge, however, is to ensure that the first increment will bolt onto (interface with) subsequent increments. You must learn enough about all of the increments that you can approximate how they will work together as an integrated product. Figure 1-2 gives a sample time line for a project using an iterative, incremental approach.

Figure 1-2. Time line of a project with iterative and incremental flow



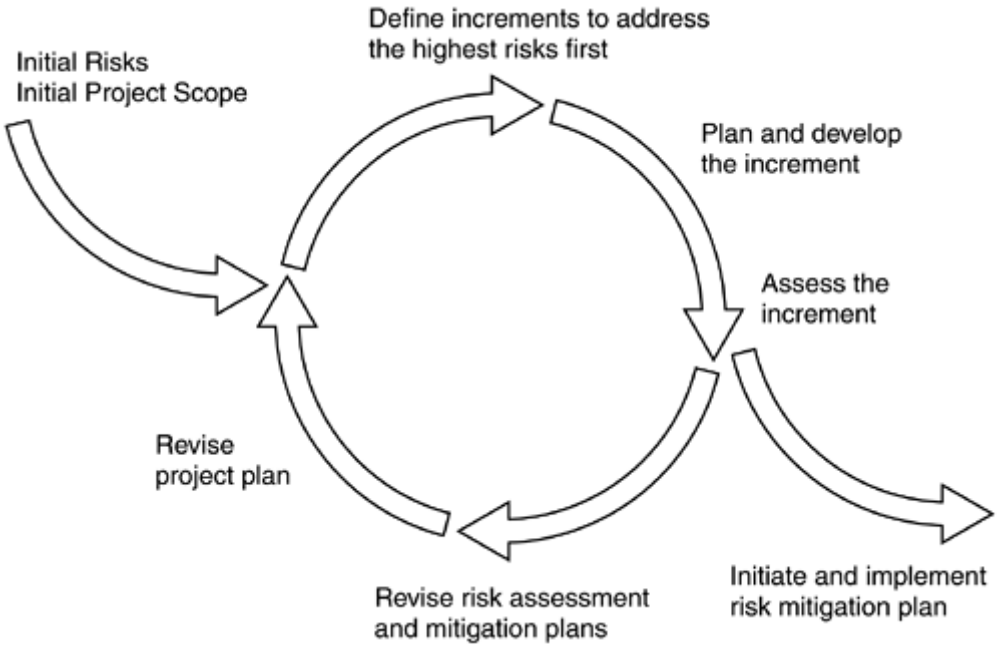
After years of applying these concepts to many different types of projects, using many different tools and languages, I have no doubt that this is the only way that software can be successfully developed today and in the future.

Risk-Based Software Development

Experience has taught me to always be a silent pessimist when I approach a new project. This idea stems from the repeated observation that something is always lurking nearby that will shift the project's course toward eventual disaster. Although this attitude might seem like a negative way to look at project development, it has saved many projects from disaster.

The project team must always be on the lookout for risks. Risks must be brought to the surface early and often. One way to do this is to extend the project development philosophy so that it is not only iterative and incremental, but also risk based. Appendix A presents project plan templates for the Unified Process. Figure 1-3 shows one possible visual representation of an iterative, incremental project framework founded on a risk-based approach.

Figure 1-3. Iterative, incremental project framework with risk mitigation



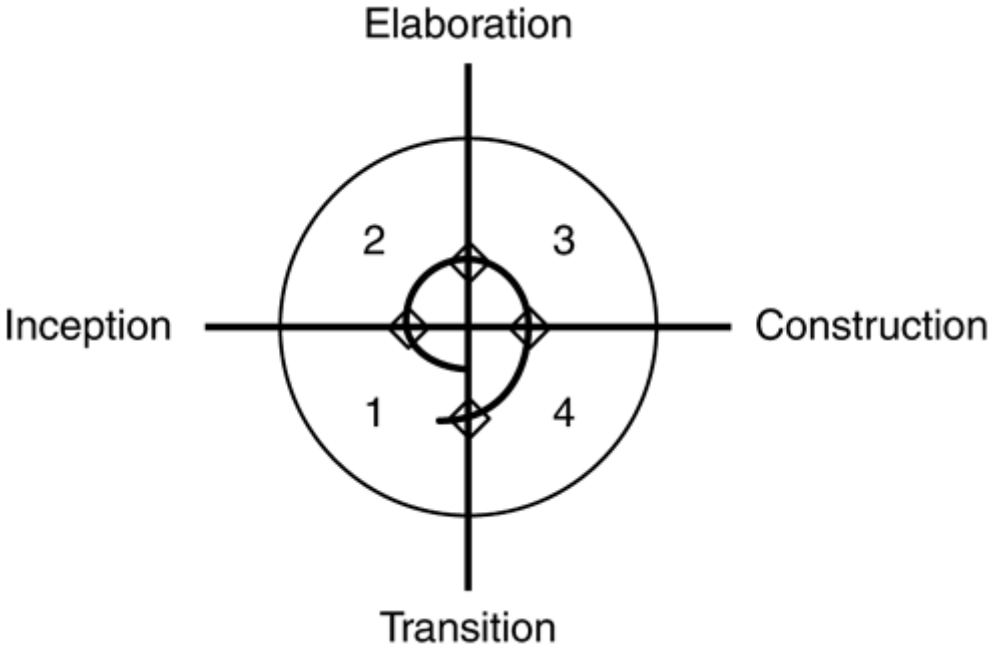
One very positive side effect of this approach is the continual improvement of the end product. In addition, risks are addressed promptly because the project components that present the greatest risk are staged first.

The Iterative Software Process Model

It helps to visualize how combining the notions of iterative and incremental development might look graphically. Figure 1-4 shows an iterative and incremental process model. This framework is sometimes called the **spiral process model** and has been popularized by many practitioners. Each of the four quadrants shown in the figure is labeled as a phase:

1. Inception
2. Elaboration
3. Construction
4. Transition

Figure 1-4. Iterative and incremental process model: One-dimensional



These four phases parallel the terminology used by the Unified Process from Rational Software. Rational's Unified Process has as its roots the Objectory Process, created by Ivar Jacobson in the 1980s.

The project begins with the Inception phase. This is a discovery phase, a time for solidifying the project vision and incorporating a clear understanding of what features the project will implement. At this point we fly by the entire project, going a mile wide and five inches deep. Out of the Inception phase we will also have determined what the use-cases are. The project will also select the architecturally significant use-cases. It is these requirements that we target in our first iteration in the Elaboration phase. The milestone reached at the end of the Inception phase is called Lifecycle Objective.

In the Elaboration phase, early requirements identified in the Inception phase are solidified, and more rigor is added to the process. The first iteration of the Elaboration phase targets requirements that will have the greatest impact on the project's understanding and development of the architecture. The deliverable from the first iteration in Elaboration is an architectural prototype. This deliverable not only provides the necessary feedback to the user?Yes, we can build software that coincides with your requirements"榊ut also it validates the proposed architecture.

Actually, there should be no "hard" architecture decisions to be made after the first iteration in the Elaboration phase. An architectural

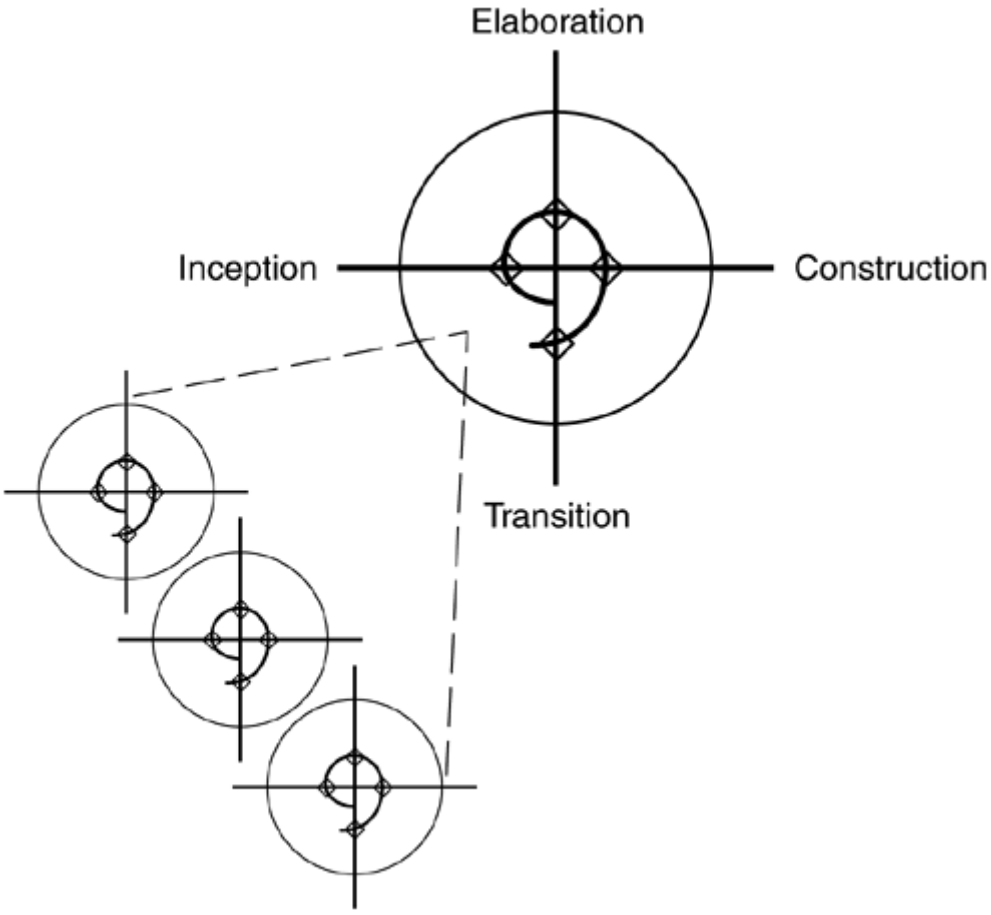
prototype evolves throughout Elaboration, as more use-cases are tackled in subsequent iterations. The milestone reached at the end of the Elaboration phase is called Lifecycle Architecture. In this book, our target is to make it through the first iteration in Elaboration and deliver an architectural prototype.

The Construction phase includes the mechanical aspects of implementing all the business rules and ensuring that subsystems integrate. This phase is, for the most part, a software manufacturing process. Absolutely no architecture surprises should arise during Construction. It is in the Construction phase that alpha releases of the system are made available to the user. Packaging, rollout, support, and training issues are also dealt with in this phase. The milestone reached at the end of the Construction phase is called Initial Operational Capability.

In the Transition phase, components produced in the Construction phase are packaged into deployable units. At this time in the project the system is typically thought to be in beta status. Some parallel operations may also be taking place alongside existing heritage applications. This phase also details the support issues that surround the application and how the project will be maintained in a production environment. The milestone reached at the end of the Transition phase is called Product Release. In the Unified Process a product release is called an *evolution* or *generation.*

Within each phase, multiple iterations typically take place. The number of iterations within each phase might vary (Figure 1-5), depending on the project's unique requirements. For projects with a higher risk factor or more unknown elements, more time will be spent learning and discovering. The resulting knowledge will have to be translated to other deliverables in a layered fashion.

Figure 1-5. Iterations within phases

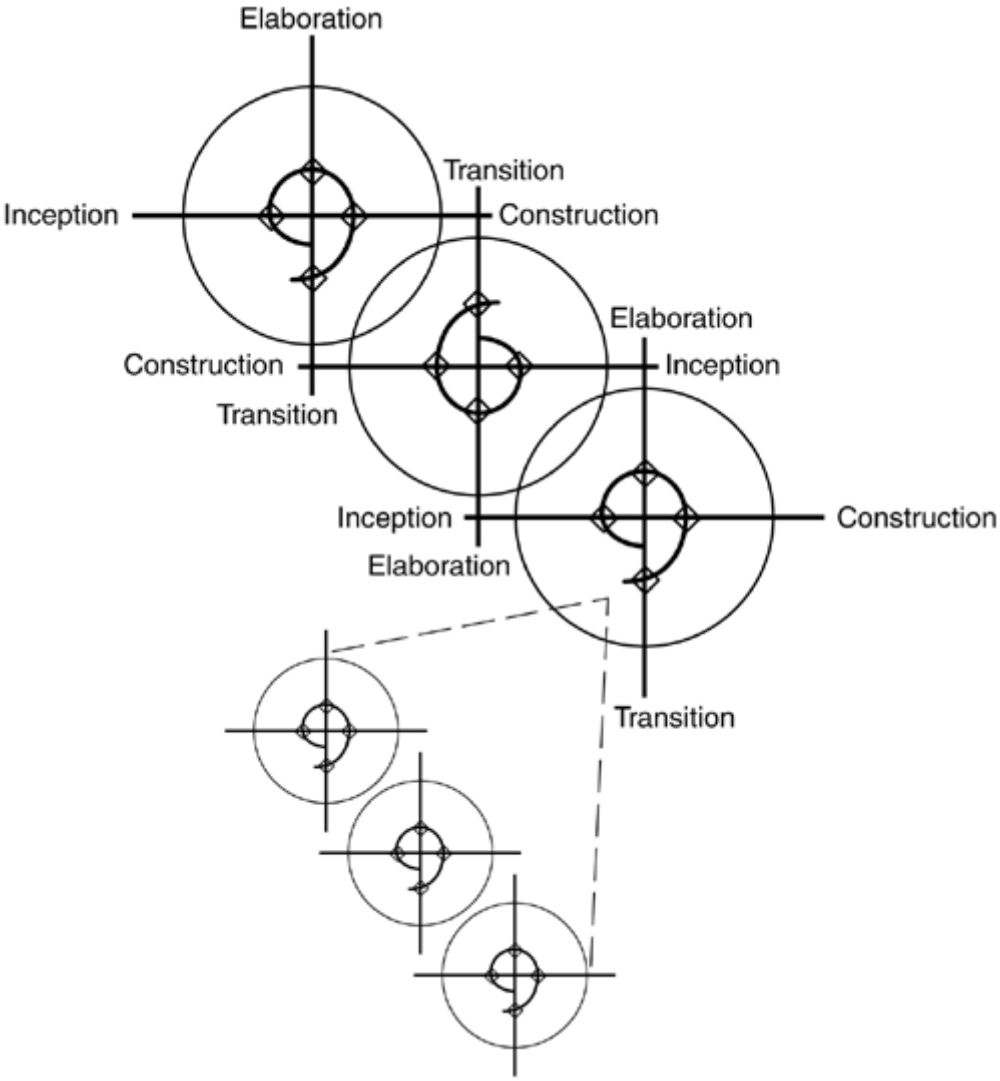


To make the project even more interesting, the activities traditionally associated with any one phase may be performed in earlier or later phases. For example, if the project has a strong, unique visual component or is breaking new ground, you might need to simulate a prototype during Inception just to generate ideas and solidify the proof of concept going forward.

Combining Iterative with Incremental: Multidimensional View

Given that an iterative process is beneficial, next we envision the previous flow imposed on an incremental delivery schedule. Figure 1-6 illustrates the iterative nature of typical software development projects and shows that different increments will be in different stages of the lifecycle.

Figure 1-6. Iterative and incremental process model: Multidimensional



Notice that each increment (the three upper spirals) is in a different phase of its evolution. Each phase (e.g., Inception) might also be in its own iteration cycle. At first glance, all this might seem overly complex. From the project manager's perspective, more balls do need to be juggled. However, from the perspectives of the user, analyst, designer, and developer, a clear demarcation exists between each increment. The reason for this approach is, once again, to lessen risk by disassembling the logical seams of the application and then attacking each increment individually.

The Synergy Process

The Synergy Process, which is based on iterative, incremental software development, combines with an implementation project plan that is your