Chapter 1

Overview

Introduction

The Unified Process fits the general definition of a process: a set of activities that a team performs to transform a set of customer requirements into a software system. However, the Unified Process is also a generic process framework that people can customize by adding and removing activities based on the particular needs and available resources for a project.

The Rational Unified Process (RUP) is an example of a specialized version of the Unified Process that adds elements to the generic framework; see Appendix A for a discussion of those elements. The discussion in the body of this book about The Internet Bookstore, an example project, represents a tailoring in the other direction: The team doing that project skipped some activities that didn't add value. This example system illustrates a key aspect of how one should put the Unified Process to work: Use those elements of it that add value for a particular project; omit those elements that don't add value. See Appendix C for an example of a streamlined process based on the principle of starting with core elements and adding other elements as necessary.

The Unified Process makes extensive use of the Unified Modeling Language (UML). At the core of the UML is the **model**, which in the context of a software development process is a simplification of reality that helps the project team understand certain aspects of the complexity inherent in software.

The UML was designed to help the participants in software development efforts build models that enable the team to visualize the system, specify the structure and behavior of that system, construct the system, and document the decisions made along the way. Many of the tasks that the Unified Process defines involve using the UML and one or more models.

See Chapter 1 of *UML Explained* for information about models and their value in software development.

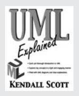
The remaining sections in this chapter describe how the Unified Process evolved, the key tenets that underlie the process (use case driven, architecture-centric, and iterative and incremental), and the vocabulary used to describe the details of the process.

History

The Unified Process has its roots in the work that Ivar Jacobson did at Ericsson in the late 1960s. Jacobson and his colleagues modeled a very large telecommunications system using layers of "blocks," with the lower layers serving as the foundation for subsystems at the higher layers. (These blocks have parallels with what are now known as components.) The team built those low-level blocks by exploring what they called traffic cases (now called use cases in the UML); all subsequent analysis, design, implementation, and test work was driven, to a greater or lesser extent, by those traffic cases. Other diagrams that are now key aspects of the UML evolved from the Ericsson work, including sequence diagrams (called object interaction diagrams at the time) and activity diagrams (which Ericsson called state transition graphs). A document called the software architecture description contained the key material that facilitated communication among developers and between developers and customers.

Jacobson went out on his own in 1987, starting a company he named Objectory AB. He and his associates spent several years developing Objectory, which was both a process and a product, as is the Rational Unified Process (discussed next). His book *Object-Oriented Software Engineering* (Addison-Wesley, 1995), which described the Objectory process in detail, was regarded as a landmark in the object-oriented (OO) community; it was the first book by a major methodologist to put forth the idea that the customer's requirements, as expressed within use cases, should be the most important driving force in software development. The genesis of the Unified Process's emphasis on architecture (see "Architecture-Centric," on the next page) were also on display in the book. The full on-line version of the process appeared in conjunction with the book in 1995.

Not long after that, Rational bought Objectory AB. Jacobson and his considerably larger set of colleagues set about expanding the areas that the Objectory process didn't address in depth, such as project management and development tools. Grady Booch and Jim Rumbaugh were already on board at Rational—Booch almost from the beginning, and Rumbaugh from late 1994—and the gentlemen who became known as the "three amigos" were among the leaders of the effort to build what eventually became the Rational Objectory Process (ROP), in parallel with their expansion of the Unified Method into what became the Unified Modeling Language.



See Chapter 1 of *UML Explained* for a brief history of the UML.

While the work on the ROP and UML was going on, Rational was busy acquiring and merging with a number of other companies that made software development tools. These tools added value to the ROP product in the form of requirements management (Requisite's tool became RequisitePro), general-purpose testing (SQA's software has been expanded into several discrete tools), and other areas such as performance testing, configuration management, and change management. In 1998, Rational changed the name of the product to the RUP; the differences between the Unified Process (which is at the conceptual core of the RUP) and the RUP as a product are described in Appendix A.

Use Case Driven

A **use case** is a sequence of actions, performed by one or more **actors** (people or non-human entities outside of the system) and by the system itself, that produces one or more results of value to one or more of the actors. One of the key aspects of the Unified Process is its use of use cases as a driving force for development. The phrase *use case driven* refers to the fact that the project team uses the use cases to drive all development work, from initial gathering and negotiation of requirements through code. (See "Requirements" later in this chapter for more on this subject.)

Use cases are highly suitable for capturing requirements and for driving analysis, design, and implementation for several reasons.

- Use cases are expressed from the perspective of the system's users, which translates into a higher comfort level for customers, as they can see themselves reflected in the use case text. It's relatively difficult for a customer to see himself or herself in the context of requirements text.
- Use cases are expressed in natural language (English or the native language of the customers). Well-written use cases are also intuitively obvious to the reader.
- Use cases offer a considerably greater ability for everyone to understand the real requirements on the system than typical requirements documents, which tend to contain a lot of ambiguous, redundant, and contradictory text. Ideally, the stakeholders should regard use cases as binding contracts between customers and developers, with all parties agreeing on the system that will be built.
- Use cases offer the ability to achieve a high degree of traceability of requirements into the models that result from ongoing development. By keeping the use cases close by at all times, the development team is always in touch with the customers' requirements.
- Use cases offer a simple way to decompose the requirements into chunks that allow for allocation of work to subteams and also facilitate project management. (See "Use Case Model" in Chapter 2 for information about breaking use cases up into UML packages.) This is *not* the same as functional decomposition, though; see *Use Case Driven Object Modeling with UML* (Rosenberg and Scott, 1999) for an explanation of the difference.

Architecture-Centric

In the context of software, the term **architecture** has different meanings depending on whom you ask. The definition in *UML Explained* is as follows:

> The fundamental organization of the system as a whole. Aspects of an architecture include static elements, dynamic elements, how those elements work together, and the overall architectural style that guides the organization of the system. Architecture also addresses issues such as performance, scalability, reuse, and economic and technological constraints.

Several books offer other useful definitions; see, for example, *Software Architecture in Practice* (Len Bass, Paul Clements, and Rick Kazman; Addison-Wesley, 1998).

The Unified Process specifies that the architecture of the system being built, as the fundamental foundation on which that system will rest, must sit at the heart of the project team's efforts to shape the system, and also that architecture, in conjunction with the use cases, must drive the exploration of all aspects of the system. You might think of the architecture as expressing the common vision that all members of the team must share in order for the resulting system to be suitably robust, flexible, expandable, and cost-effective.

In the context of the process, architecture is primarily specified in terms of views of six models. (See "The Five Workflows" later in this chapter for brief descriptions of these models.) These views reflect the "architecturally significant" elements of those models; taken together, the views form the **architecture description**. The project team initializes the architecture description early, then expands and refines it during virtually all the activities of the project.

The following subsections discuss the key reasons why architecture is so important to the Unified Process.

Understanding the Big Picture

The tools and techniques available to developers for building software are growing increasingly powerful. For better or worse, though, software itself, especially with its new focus on distributed computing, is getting considerably more complex as well, and there aren't any indications that the tools and techniques will "catch up" any time soon. Also, customers' attention spans are becoming shorter and shorter as their demands on development teams grow more sophisticated. The result is that it's very difficult for all but a few especially gifted people to understand—*really* understand—most software systems to any meaningful extent. The architecture description is meant, first and foremost, to facilitate an understanding of the architecture of the system being built. Rigorous modeling, and careful attention to the readability of the associated UML diagrams and supporting text, will go a long way toward turning the architecture description into the fulcrum for increased understanding of the “big picture” of the new system.

Organizing the Development Effort

A sound architecture explicitly defines discrete chunks of the system, as well as the interfaces among the various parts of the system. It also makes effective use of one or more architectural patterns, which help shape the development effort on various levels. (Client/server, three-tier, and N-tier are all examples of well-known architectural patterns. Other patterns focus on things like object request brokers [ORBs], which sit at the center of systems that use distributed components, and virtual machines, such as the one on top of which Java runs.) By using this aspect of architecture effectively, the project team can increase the chances that communication across subteams will add value to the effort.

Facilitating the Possibilities for Reuse

One of the key tenets of component-based development (CBD) is the idea that components should be usable, with a relative minimum of customization, in a variety of contexts. A well-constructed software architecture offers solid “scaffolding” on which components can reside and work gracefully with each other, while making it easy for teams building other systems to identify opportunities for possible reuse of any or all of those components. The bottom line is that the less time a team has to spend focusing on building new components, the more time it can spend on understanding the customers’ problems and modeling the solutions.

Evolving the System

Maintaining and enhancing a system tends to occupy more time over the life of that system than it took to build it in the first place. When development projects find themselves operating in mythical “Internet time,” with technologies evolving faster and business models changing more frequently than ever before, there’s no question that a system of any size and complexity will be subject to evolutionary changes of a healthy magnitude. Having a solid architecture in place offers a set of essential reference points on which future development work can rely. An architecture that's been built such that changes in one part of the system almost never have adverse effects on other parts of the system also greatly enhances team members' ability to evolve the system effectively and efficiently.

Guiding the Use Cases

In one sense, use cases drive the architecture of a software system, since the use cases do drive all of the development effort. In another sense, however, the architecture guides the selection and exploration of use cases. Decisions that architects must make about things like middleware, system software, legacy systems, and so forth, have a strong influence on the choice of which use cases the team focuses on at what point in the project. The basic idea, then, is to focus on those use cases that will add value to the architecture, which in turn helps shape the content of those use cases and the nature of the work involved in developing the system from them.

Iterative and Incremental

The third fundamental tenet of the Unified Process is its *iterative and incremental* nature. An **iteration** is a mini-project that results in a version of the system that will be released internally or externally. This version is supposed to offer incremental improvement over the previous version, which is why the result of an iteration is called an **increment**.

The section "Iterations and Increments," which appears later in this chapter, describes how iterations and increments fit into the larger context of the overall process. Meanwhile, the following subsections describe the advantages of iterative and incremental development.

Logical Progress Toward a Robust Architecture

An earlier section, "Architecture-Centric," describes the central place of architecture in the Unified Process. The process specifies how the project team should focus on particular aspects of the architecture during each of the iterations of the system. During early iterations, the team puts together a candidate architecture that offers the beginnings of a solid foundation; later iterations find the team expanding the vision of the full architecture, which in turn influences most, and in some cases all, of the development tasks being performed as part of a given iteration. Building the architecture in an iterative and incremental fashion enables the team to make necessary major changes early in the process at considerably less cost than they would inflict later in the project.

Dealing With Ongoing Changes in Requirements

Processes based on the waterfall approach, which dictates that all of the requirements be gathered and analyzed before design starts, face what now seems to be an inevitable problem: Requirements tend to be unstable. Also, customers have difficulty envisioning a system when all they have is documentation. The Unified Process advocates breaking the system down into **builds**, where each build is a working version of some meaningful chunk of the full system. By focusing on bounded sets of use cases and making effective use of prototypes, the project team and the customers can negotiate requirements on an ongoing basis, thus reducing the (often very large) risk associated with trying to specify all of the requirements up front. One of the reviewers of the manuscript for this book indicated that his company practices "ruthless prioritization," which involves dealing with changing requirements by aggressively identifying priorities and eliminating lower-priority features from consideration.

Greater Flexibility to Change the Plan

Since each iteration is a mini-project, the project team addresses, to some extent, all the risks associated with the project as a whole each time it builds an increment of the system. As risks become greater, as delays occur, and as the environment becomes more unstable, the team is able to make necessary adjustments on a relatively small scale and propagate those adjustments across the entire project. During the postmortem for each iteration, the project leaders can decide whether the iteration was a success and change the iteration plan as appropriate before work proceeds with the next iteration. The goal is to isolate problems within iterations and deal with them on a relatively small scale, rather than allowing them to spread.

Continuous Integration

Each increment brings a combination of new features and improved functionality to the system. This enables all the stakeholders to measure the progress of the project toward specific goals, rather than toward more abstract and general requirements. By continually integrating new increments, the development team is also able to isolate problems that it might bring to the system and address those problems in ways that don't disrupt the integrity of the working system. This kind of setup makes it easier for the team to go as far as throwing a particular increment away and starting over, since the process gives it the ability to define iterations that take less time to perform.

Early Understanding

Each of the activities that the team performs during an iteration is straightforwardly defined, as is the sequence of activities within each workflow and across workflows. The process is designed to enable reliance on things like ongoing mentoring, rather than forcing people to go through extensive training before becoming productive members of the team. Well-defined iterations allow room to experiment and make mistakes, because those mistakes will be isolated such that their impact on schedule and budget can be minimized. As work proceeds, the team can leverage its understanding of what it's trying to build and the associated risks, thus building momentum, which in turn enables the team to make continuous improvements in the way it goes about its tasks.

Ongoing Focus on Risk

Perhaps the most important advantage that iterative and incremental development, as defined by the Unified Process, brings to the table is the project team's ability to focus its efforts on addressing the most critical risks early in the life cycle. The team has a mandate to organize iterations based on addressing risks on an ongoing basis; the goal is to mitigate risks to the greatest extent possible during each iteration, so each iteration poses fewer risks of less importance than its predecessors.

Various people have come up with many different ways to categorize risks to software development projects. See *UML Distilled* (Martin Fowler with Kendall Scott; Addison-Wesley, 1999) for one view of risks. Three categories of risks useful in discussing the Unified Process are described next.

- **Technical risks** are those associated with the various technologies that will come into play during the project and with issues such as performance and the "-ilities" (reliability, scalability, and so forth). For example, if the system will use Enterprise Java Beans (EJBs) in the context of the Common Object Request Broker Architecture (CORBA), the project team must solve a number of potential technical problems along the way to building a system that will perform acceptably. The process doesn't specifically address technical risks; however, the emphasis on architecture (see below) reflects the principle that the team should address technical risks early, before coding starts.
- **Architectural risks** are those associated with the ability of the architecture to serve as a strong foundation of the system and also be sufficiently resilient and adaptable to the addition of features in future releases of the system. Risks associated with "make versus buy" decisions are also part of this category. The process addresses architectural risks by defining activities that involve analyzing, designing, and implementing the architecture, and by defining a number of other activities that include keeping the architecture description up to date so it can assume its place front and center within the development effort.
- **Requirements risk** is the risk of not building the right system—the system that the customers are paying for—by not understanding the requirements and not using associated use cases to drive development. The process addresses requirements risk with activities specifically defined to facilitate the discovery and negotiation of requirements and with its premise that use cases should drive all aspects of development.

The Four Phases

The life of a software system can be represented as a series of **cycles**. A cycle ends with the release of a version of the system to customers.

Within the Unified Process, each cycle contains four phases. A **phase** is simply the span of time between two **major milestones**, points at which managers make important decisions about whether to proceed with development and, if so, what's required concerning project scope, budget, and schedule.

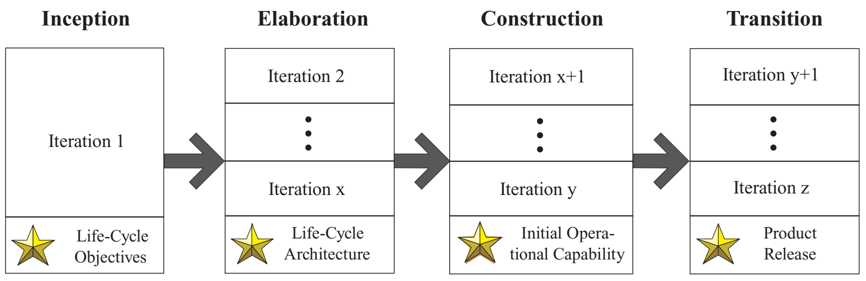


Figure 1-1: Phases and Major Milestones

Figure 1-1 shows the phases and major milestones of the Unified Process. In it, you can see that each phase contains one or more iterations. We'll explore the concept of iterations in the section "Iterations and Increments" later in this chapter.

The following subsections describe the key aspects of each of these phases.

Inception

The primary goal of the **Inception phase** is to establish the case for the viability of the proposed system.

The tasks that a project team performs during Inception include the following:

- Defining the scope of the system (that is, what's in and what's out)
- Outlining a **candidate architecture**, which is made up of initial versions of six different models
- Identifying critical risks and determining when and how the project will address them
- Starting to make the business case that the project is worth doing, based on initial estimates of cost, effort, schedule, and product quality

The concept of candidate architecture is discussed in the section "Architecture-Centric" later in this chapter. The six models are covered in the next major section of this chapter, "The Five Workflows."

The major milestone associated with the Inception phase is called **Life-Cycle Objectives**. The indications that the project has reached this milestone include the following:

- The major stakeholders agree on the scope of the proposed system.
- The candidate architecture clearly addresses a set of critical high-level requirements.
- The business case for the project is strong enough to justify a green light for continued development.

Chapter 7 describes the details of the Inception phase.

Elaboration

The primary goal of the **Elaboration phase** is to establish the ability to build the new system given the financial constraints, schedule constraints, and other kinds of constraints that the development project faces.

The tasks that a project team performs during Elaboration include the following:

- Capturing a healthy majority of the remaining functional requirements
- Expanding the candidate architecture into a full **architectural baseline**, which is an internal release of the system focused on describing the architecture
- Addressing significant risks on an ongoing basis
- Finalizing the business case for the project and preparing a project plan that contains sufficient detail to guide the next phase of the project (Construction)

The architectural baseline contains expanded versions of the six models initialized during the Inception phase.

The major milestone associated with the Elaboration phase is called **Life-Cycle Architecture**. The indications that the project has reached this milestone include the following:

- Most of the functional requirements for the new system have been captured in the use case model.
- The architectural baseline is a small, skinny system that will serve as a solid foundation for ongoing development.
- The business case has received a green light, and the project team has an initial project plan that describes how the Construction phase will proceed.

The use case model is described in the upcoming section "The Five Workflows." Risks are discussed in the section "Iterations and Increments" later in this chapter.

Chapter 8 describes the details of the Elaboration phase.

Construction

The primary goal of the **Construction phase** is to build a system capable of operating successfully in beta customer environments.

During Construction, the project team performs tasks that involve building the system iteratively and incrementally (see "Iterations and Increments" later in this chapter), making sure that the viability of the system is always evident in executable form.

The major milestone associated with the Construction phase is called **Initial Operational Capability**. The project has reached this milestone if a set of beta customers has a more or less fully operational system in their hands.

Chapter 9 describes the details of the Construction phase.

Transition

The primary goal of the **Transition phase** is to roll out the fully functional system to customers.

During Transition, the project team focuses on correcting defects and modifying the system to correct previously unidentified problems.

The major milestone associated with the Transition phase is called **Product Release**.

Chapter 10 describes the details of the Transition phase.

The Five Workflows

Within the Unified Process, five **workflows** cut across the set of four phases: Requirements, Analysis, Design, Implementation, and Test. Each workflow is a set of activities that various project workers perform.

The following subsections provide brief overviews of these workflows.

Requirements

The primary activities of the **Requirements workflow** are aimed at building the use case model, which captures the functional requirements of the system being defined. This model helps the project stakeholders reach agreement on the capabilities of the system and the conditions to which it must conform.

The use case model also serves as the foundation for all other development work. Figure 1-2 shows how the use case model influences the other five models discussed in the subsequent subsections.

Chapter 2 discusses the key aspects of the Requirements workflow. Chapters 7, 8, and 9 describe how this workflow cuts across the Inception, Elaboration, and Construction phases, respectively.

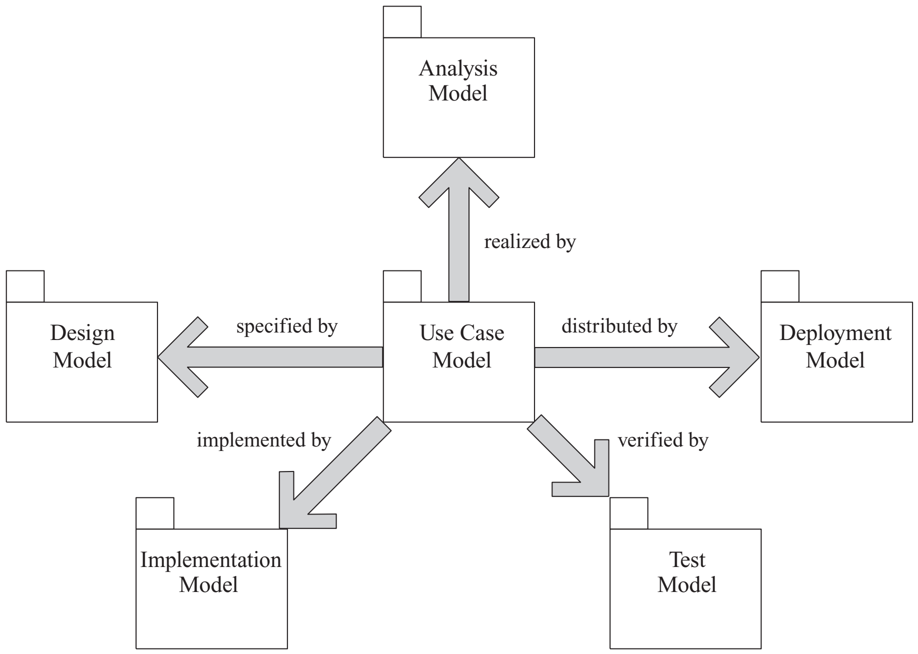


Figure 1-2: The Six Basic Unified Process Models

Analysis

The primary activities of the **Analysis workflow** are aimed at building the analysis model, which helps the developers refine and structure the functional requirements captured within the use case model. This model contains realizations of use cases that lend themselves to design and implementation work better than the use cases.

Chapter 3 discusses the key aspects of the Analysis workflow. Chapters 7, 8, and 9 describe how this workflow cuts across the Inception, Elaboration, and Construction phases, respectively.

Design

The primary activities of the **Design workflow** are aimed at building the design model, which describes the physical realizations of the use cases from the use case model, and also the contents of the analysis model. The design model serves as an abstraction of the implementation model (see the next subsection).

The Design workflow also focuses on the deployment model, which defines the physical organization of the system in terms of computational nodes.

Chapter 4 discusses the key aspects of the Design workflow. Chapters 7, 8, and 9 describe how this workflow cuts across the Inception, Elaboration, and Construction phases, respectively.

Implementation

The primary activities of the **Implementation workflow** are aimed at building the implementation model, which describes how the elements of the design model are packaged into software components, such as source code files, dynamic link libraries (DLLs), and EJBs.

Chapter 5 discusses the key aspects of the Implementation workflow. Chapters 8 and 9 describe how this workflow cuts across the Elaboration and Construction phases, respectively.

Test

The primary activities of the **Test workflow** are aimed at building the test model, which describes how integration and system tests will exercise executable components from the implementation model. The test model also describes how the team will perform those tests as well as unit tests.

The test model contains test cases that are often derived directly from use cases. Testers perform black-box testing using the original use case text, and white-box testing of the realizations of those use cases, as specified within the analysis model. The test model also contains the results of all levels of testing.

Chapter 6 discusses the key aspects of the Test workflow. Chapters 8 and 9 describe how this workflow cuts across the Elaboration and Construction phases, respectively.

Iterations and Increments

As mentioned in "The Four Phases," each of the Unified Process's phases is divided into iterations. An **iteration** is simply a mini-project that's part of a phase.

A typical iteration crosses all five of the workflows discussed in the previous section, to a greater or lesser extent. For instance, an iteration during the Elaboration phase might focus heavily on activities of the Requirements and Analysis workflows, whereas an iteration during Construction is more likely to involve Design, Implementation, and Test activities. (Chapters 7 through 9 discuss the details of these crossovers.)

Each iteration results in an **increment**. This is a release of the system that contains added or improved functionality compared with the previous release.

Figure 1-3 shows the essence of the iterative and incremental approach to software development.

Using an iterative and incremental approach, a project team starts the development process by evaluating the relevant risks, including those associated with requirements, skills, technology, and politics, and by ensuring that the scope of the project is defined to everyone's satisfaction (see "Elaboration"). Then the team follows these steps:

1. Define the first iteration, addressing the most critical and difficult risks. (In other words, do the hard stuff first.)
2. Map out a plan for the iteration to a suitable level of detail.

Chapter 1.fm Page 17 Tuesday, October 30, 2001 2:49 PM

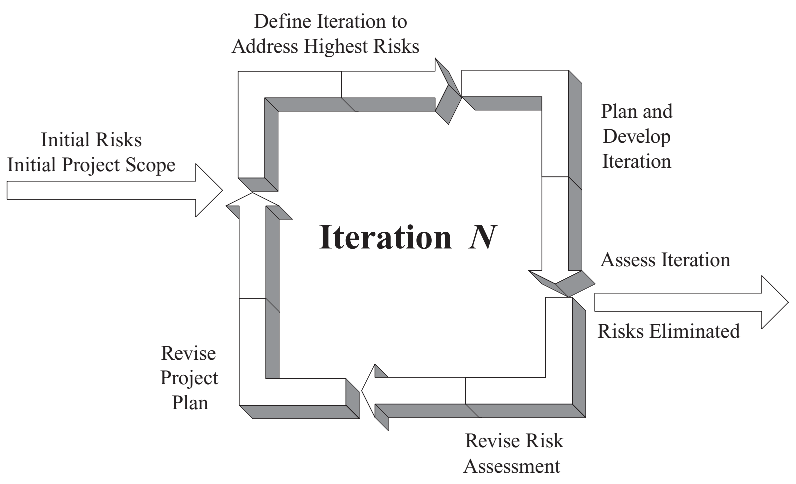


Figure 1-3: Iterative and Incremental Development

3. Perform the appropriate activities; for the Unified Process, these are activities associated with the Requirements, Analysis, Design, Implementation, and Test workflows.
4. Do a postmortem on the increment that results from the iteration.
5. Discard the risks that the increment has sufficiently addressed. Then update the ongoing risk list.
6. Revise the overall project plan in response to the relative success or failure of the iteration.
7. Proceed with the next iteration.

Iterations build the six models increment by increment. At the end of each iteration, the full set of models that represents the system is in a particular state; this is the architectural baseline.

Artifacts, Workers, and Activities

The following subsections describe three key elements of each of the workflows within the Unified Process: artifacts, workers, and activities.

Artifacts

Within the Unified Process, an **artifact** is any meaningful internal or deliverable chunk of information that plays a role in the development of the system. This book focuses on engineering artifacts, which include things like models, the user-interface prototype, and test evaluations. These artifacts are defined within the workflow chapters, and they're also discussed as deliverables of the phases in which they come into play. Management artifacts, such as the business case and the project plan, are also discussed in the contexts of the five workflows and the four phases. One of the underlying premises of the Unified Process is that a system is not considered fully deliverable until the appropriate artifacts, whether for internal use only or for customers, are reasonably complete.

Workers

The Unified Process defines a **worker** as a role that an individual may play on the project. (The primary difference between a worker and an actor is that an actor is on the outside looking in, whereas a worker is on the inside, perhaps looking out, perhaps not. Also, actors have operational or usage relationships with the system, whereas workers are participants in the development of the system.) Workers produce artifacts, either as individuals or as part of subteams or the team as a whole. One thing to remember is that one person can perform as more than one worker over the course of the project; for example, an analyst may discover use cases and write text for them as well.

Activities

Each workflow comprises several activities. In the context of a workflow, an **activity** is a task that a worker performs in order to produce an artifact. The activities described in this book range from high-level exploration of the concepts and things of interest to the customers (Build the Domain Model) to highly detailed work related to the physical system (Implement a Class).