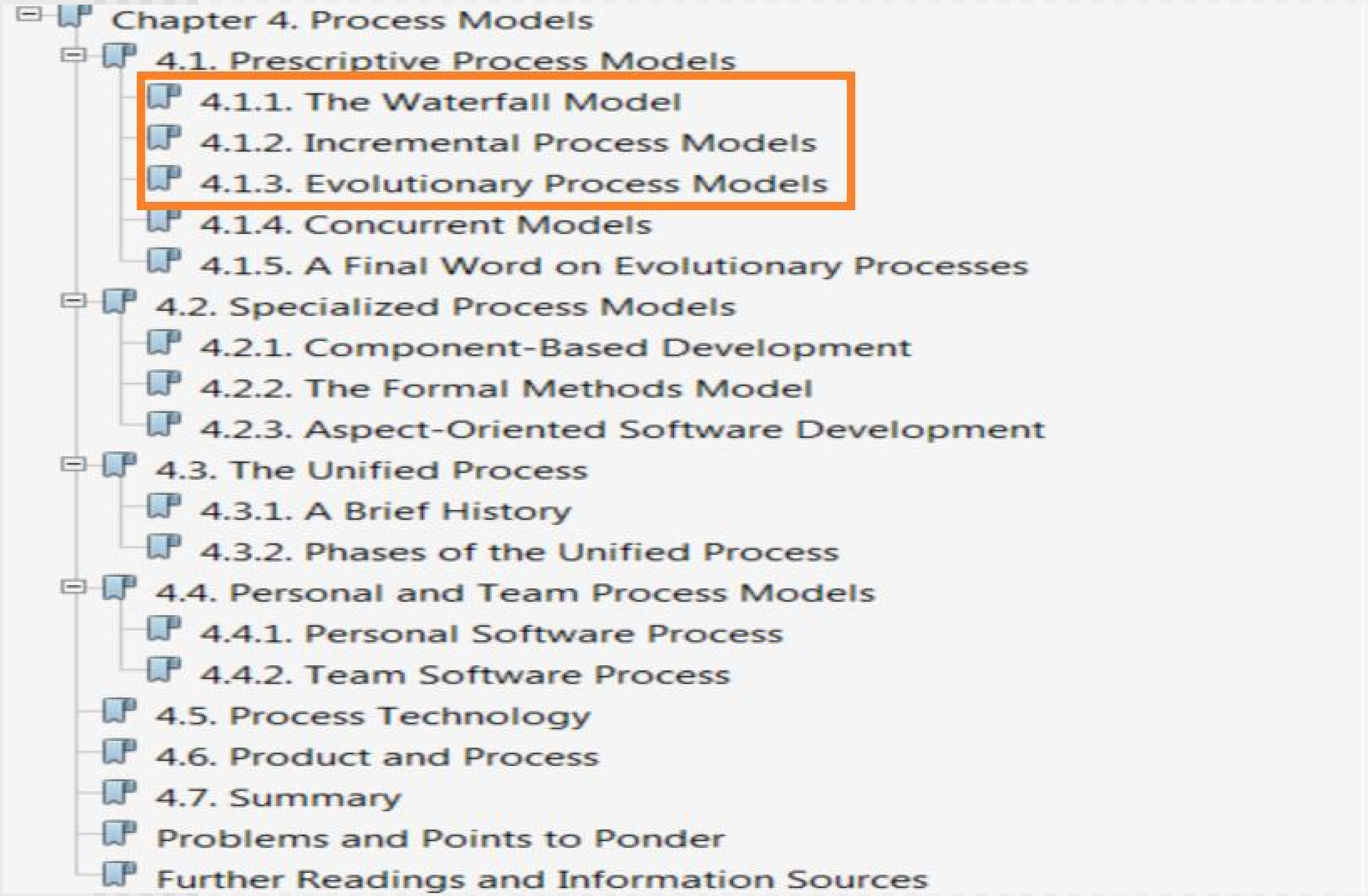
Chapter 4 **PROCESS MODELS**



Chapter 4 **PROCESS MODELS**

A process model provides a specific roadmap for software engineering work. It defines the flow of all activities, actions and tasks, the degree of iteration, the work products, and the organization of the work that must be done.

Software engineers and their managers adapt a process model to their needs and then follow it. In addition, the people who have requested the software have a role to play in the process of defining, building, and testing it.

4.1 PRESCRIPTIVE PROCESS MODELS

A prescriptive process model strives for structure and order in software development.

We call them “prescriptive” because they prescribe a set of process elements—framework activities, software engineering

Chapter 4 PROCESS MODELS

actions, tasks, work products,quality assurance, and change control mechanisms for each project. Each process model also prescribes a process flow (also called a work flow)—that is, the manner in which the process elements are interrelated to one another.

All software process models can accommodate the generic framework activities described in Chapters 2 and 3.

4.1.1 The Waterfall Model

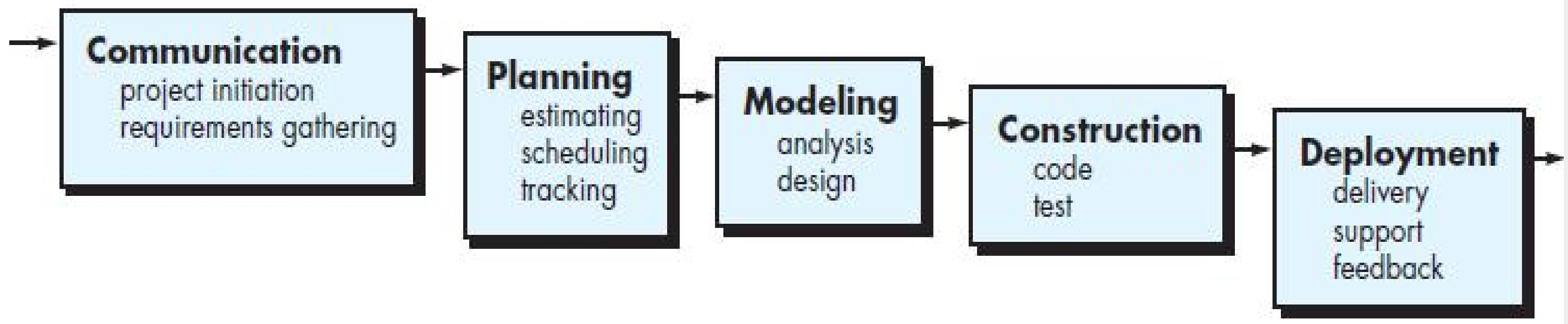
There are times when the requirements for a problem are well understood—when work flows from communication through deployment in a reasonably linear fashion. This situation is sometimes encountered when well-defined adaptations or enhancements to an existing system must be made (e.g., an adaptation to accounting software that has been mandated

Chapter 4 PROCESS MODELS

because of changes to government regulations). It may also occur in a limited number of new development efforts, but only when requirements are well defined and reasonably stable.

The waterfall model, sometimes called the classic life cycle , suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software (Figure 4.1).

FIGURE 4.1 The waterfall model



Chapter 4 PROCESS MODELS

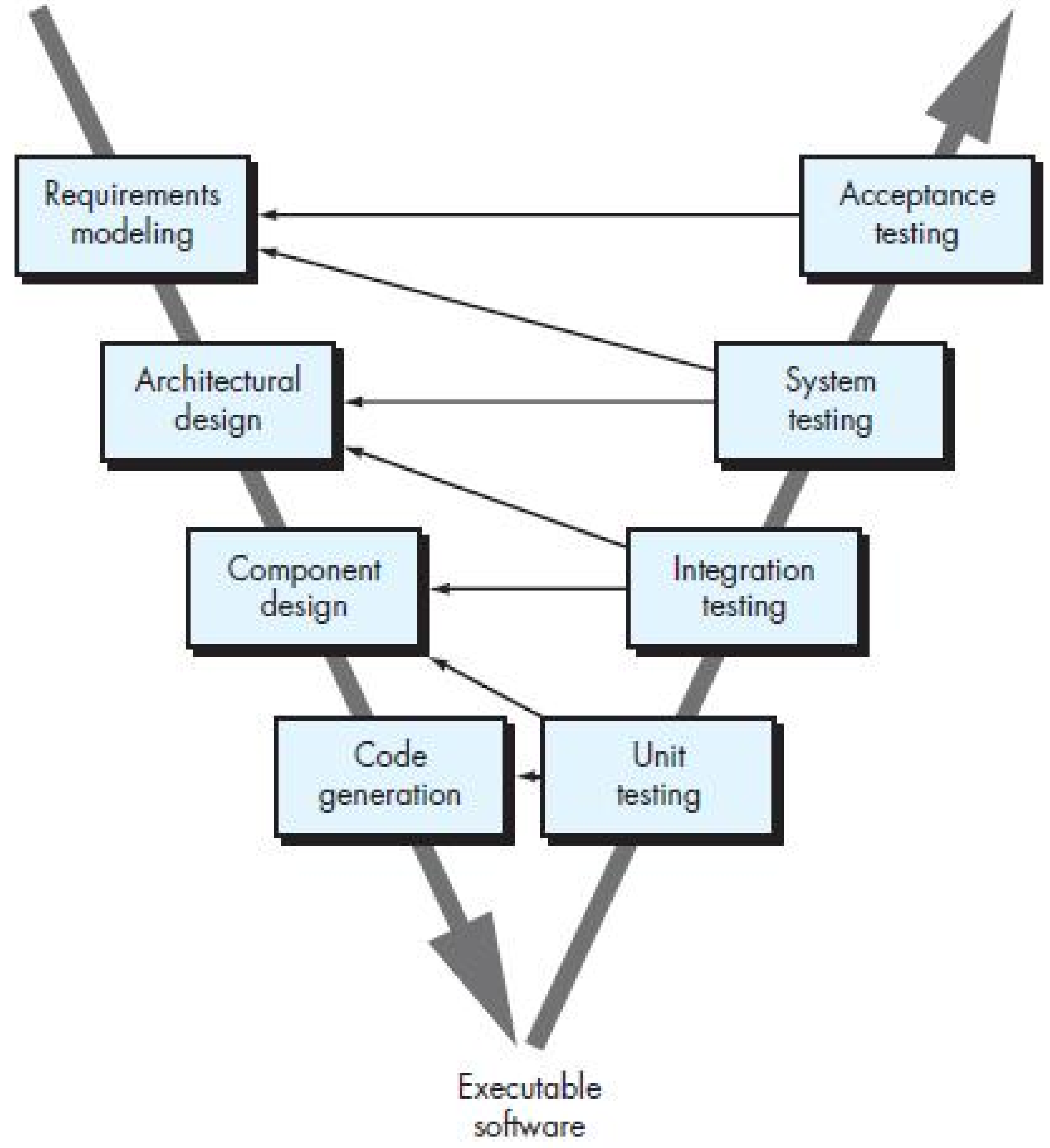
A variation in the representation of the waterfall model is called the V-model. Represented in Figure 4.2 , the V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities.

The waterfall model is the oldest paradigm for software engineering. However, over the past four decades, criticism of this process model has caused even ardent supporters to question its efficacy. Among the problems that are sometimes encountered when the waterfall model is applied are:

① **Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.**

FIGURE 4.2

The V-model



Chapter 4 PROCESS MODELS

② **It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.**

③ **The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous.**

In an interesting analysis of actual projects, Bradac found that the linear nature of the classic life cycle leads to "blocking states" in which some project team members must wait for other members of the team to complete dependent tasks.

Today, software work is fast paced and subject to a never-ending stream of changes (to features, functions, and information

Chapter 4 PROCESS MODELS

content). The waterfall model is often inappropriate for such work. However, it can serve as a useful process model in situations where requirements are fixed and work is to proceed to completion in a linear manner.

4.1.2 Incremental Process Models

There are many situations in which initial software requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process. In addition, there may be a compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later software releases. In such cases, you can choose a process model that is designed to produce the software in increments.

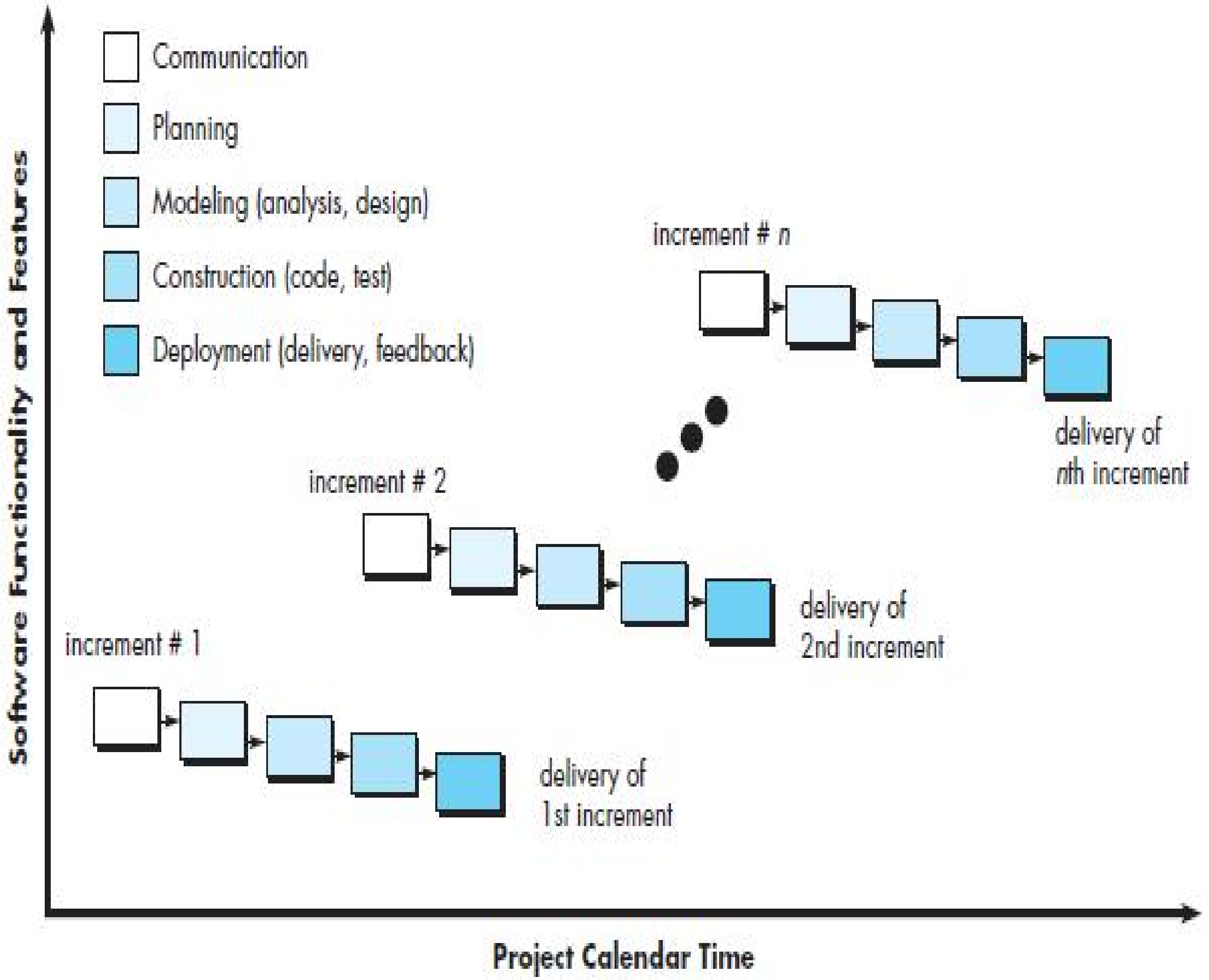
Chapter 4 PROCESS MODELS

The incremental model combines the elements' linear and parallel process flows discussed in Chapter 3. Referring to Figure 4.3 , the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software.

When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation). As a result of use and/or evaluation, a plan is developed for the next increment.

FIGURE 4.3

The incremental model


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

Chapter 4 PROCESS MODELS

4.1.3 Evolutionary Process Models

1. Software, like all complex systems, evolves over a period of time. Business and product requirements often change as development proceeds, making a straight line path to an end product unrealistic;

2.tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure;

3.a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.

Evolutionary models are iterative. They are characterized in a manner that enables you to develop increasingly more complete versions of the software. In the paragraphs that follow, we present two common evolutionary process models.

Chapter 4 PROCESS MODELS

Prototyping. Often, a customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these, and many other situations, a prototyping paradigm may offer the best approach.

The prototyping paradigm assists you and other stakeholders to better understand what is to be built when requirements are fuzzy.

The prototyping paradigm (Figure 4.4) begins with communication. You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. A prototyping iteration is planned quickly, and modeling (in the

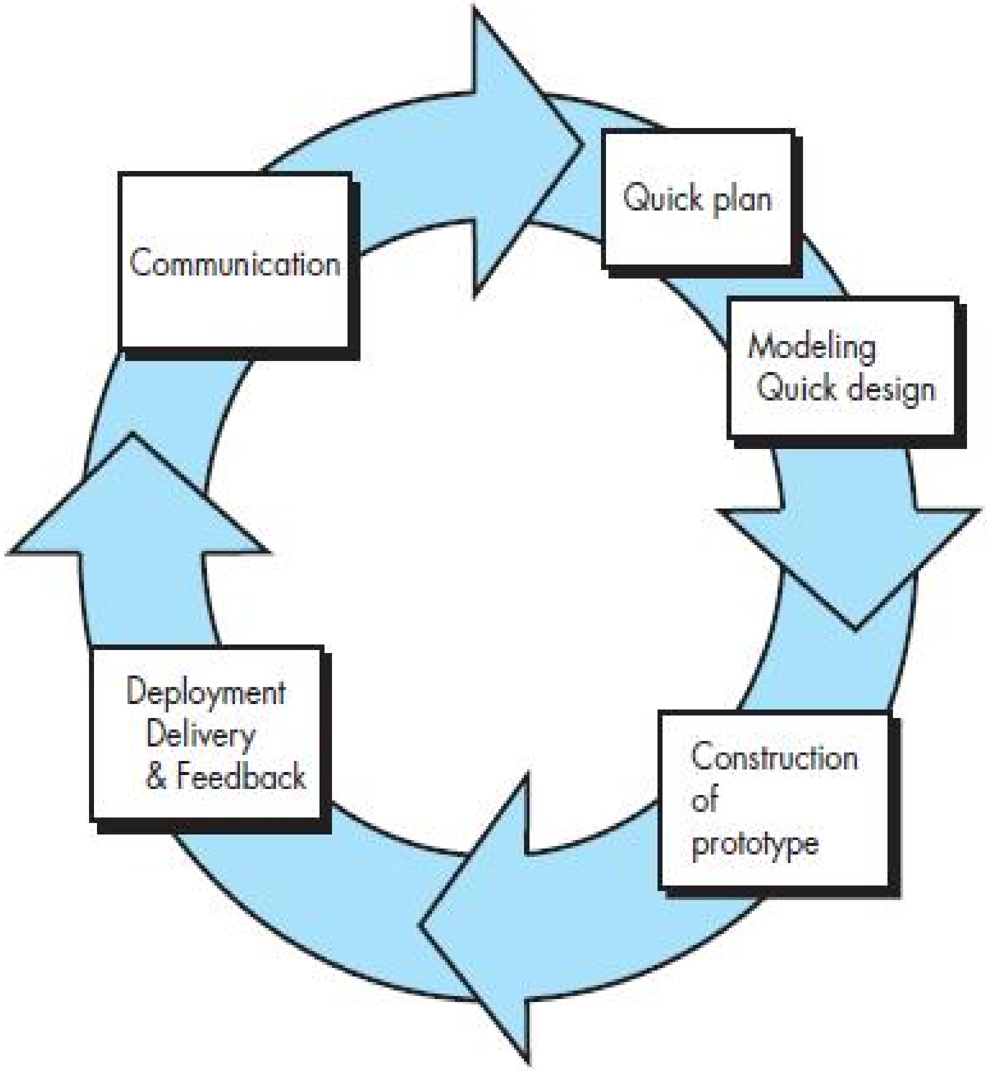
Chapter 4 PROCESS MODELS

form of a "quick design") occurs. A quick design focuses on a representation of those aspects of the software that will be visible to end users(e.g., human interface layout or output display formats). The quick design leads to the construction of a prototype. The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements. Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done.

Ideally, the prototype serves as a mechanism for identifying software requirements. If a working prototype is to be built, you can make use of existing program fragments or apply tools that enable working programs to be generated quickly.

FIGURE 4.4

The prototyping paradigm



Chapter 4 PROCESS MODELS

The prototype can serve as "the first system." The one that Brooks recommends you throw away. But this may be an idealized view. Although some prototypes are built as "throwaways," others are evolutionary in the sense that the prototype slowly evolves into the actual system.

Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately. Yet, prototyping can be problematic for the following reasons:

① **Stakeholders see what appears to be a working version of the software,unaware that the prototype is held together, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability.**

Chapter 4 PROCESS MODELS

② **As a software engineer, you often make implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability. After a time, you may become comfortable with these choices and forget all the reasons why they were inappropriate.**

Although problems can occur, prototyping can be an effective paradigm for software engineering. All stakeholders should agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded (at least in part), and the actual software is engineered with an eye toward quality.

Give an example of Marriage registration system

Chapter 4 PROCESS MODELS

The Spiral Model. **Originally proposed by Barry Boehm, the spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software.**

Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a set of framework activities defined by the software engineering team. For illustrative purposes, we use the generic framework activities discussed earlier. Each of the framework activities represent one segment of the spiral path

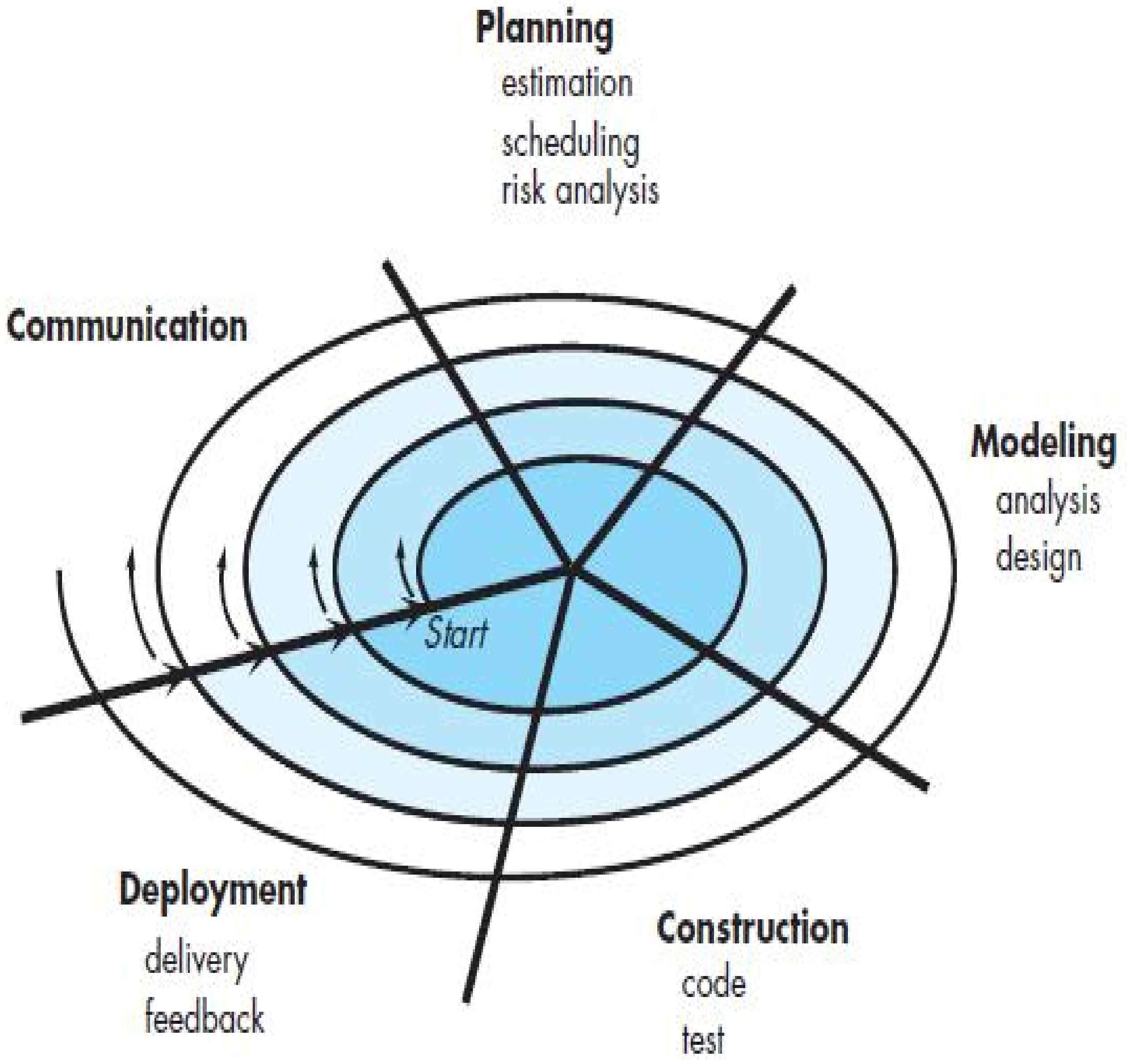
Chapter 4 PROCESS MODELS

illustrated in Figure 4.5 . As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center. Risk is considered as each revolution is made. *Anchor point milestones* —a combination of work products and conditions that are attained along the path of the spiral— are noted for each evolutionary pass.

The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager

FIGURE 4.5

A typical
spiral model



Chapter 4 PROCESS MODELS

adjusts the planned number of iterations required to complete the software.

Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the computer software. Therefore, the first circuit around the spiral might represent a "concept development project" that starts at the core of the spiral and continues for multiple iterations until concept development is complete. If the concept is to be developed into an actual product, the process proceeds outward on the spiral and a "new product development project" commences. The new product will evolve through a number of iterations around the spiral. Later, a circuit around the spiral might be used to represent a "product enhancement project."

Chapter 4 PROCESS MODELS

The spiral model is a realistic approach to the development of large-scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level. The spiral model demands a direct consideration of technical risks at all stages of the project.

But like other paradigms,It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable. It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.

Chapter 4 PROCESS MODELS

4.1.4 Concurrent Models(Read it by yourselves)

4.1.5 A Final Word on Evolutionary Processes

We have already noted that modern computer software is characterized by continual change, by very tight time lines, and by an emphatic need for customer–user satisfaction. In many cases, time-to-market is the most important management requirement. If a market window is missed, the software project itself may be meaningless.

Evolutionary process models were conceived to address these issues, and yet, they too have weaknesses, such as little focusing on development speed,flexibility, and extensibility.

The intent of evolutionary models is to develop high-quality software in an iterative or incremental manner. However, it is

Chapter 4 PROCESS MODELS

possible to use an evolutionary process to emphasize flexibility, extensibility, and speed of development. The challenge for software teams and their managers is to establish a proper balance between these critical project and product parameters and customer satisfaction.

4.2 SPECIALIZED PROCESS MODELS

Specialized process models take on many of the characteristics of one or more of the traditional models presented in the preceding sections. However, these models tend to be applied when a specialized or narrowly defined software engineering approach is chosen.

4.2.1 Component-Based Development

Commercial off-the-shelf (COTS) software components, developed

Chapter 4 PROCESS MODELS

by vendors who offer them as products, provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built.

The component-based development model **incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software. However, the component based development model comprises applications from prepackaged software components.**

Modeling and construction activities begin with the identification of candidate components. These components can be designed as either conventional software modules or object-oriented classes or packages of classes. Regardless of the technology that is used to create the components, the component-based development model incorporates the following steps (implemented using an evolutionary approach):

Chapter 4 PROCESS MODELS

① Available component-based products are researched and evaluated for the application domain in question.

② Component integration issues are considered.

③ A software architecture is designed to accommodate the components.

④ Components are integrated into the architecture.

⑤ Comprehensive testing is conducted to ensure proper functionality.

The component-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits including a reduction in development cycle time and a reduction in project cost if component reuse becomes part of your organization's culture. Component-based development is discussed in more detail in Chapter 14.

Chapter 4 PROCESS MODELS

4.2.2 The Formal Methods Model

The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software. Formal methods enable you to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation. A variation on this approach, called cleanroom software engineering, is currently applied by some software development organizations.

When formal methods (Appendix 3 in the textbook) are used during development, they provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms. Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily—not through adhoc review, but through the application of mathematical analysis.

Chapter 4 PROCESS MODELS

When formal methods are used during design, they serve as a basis for program verification and therefore enable you to discover and correct errors that might otherwise go undetected.

Although not a mainstream approach, the formal methods model offers the promise of defect-free software. Yet, concern about its applicability in a business environment has been voiced:

- The development of formal models is currently quite time consuming and expensive.
- Because few software developers have the necessary background to apply formal methods, extensive training is required.
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

Chapter 4 PROCESS MODELS

These concerns notwithstanding, the formal methods approach has gained adherents among software developers who must build safety-critical software(e.g., developers of aircraft avionics and medical devices) and among developers that would suffer severe economic hardship should software errors occur.

4.2.3 Aspect-Oriented Software Development

A distinct aspect-oriented process has not yet matured. A detailed discussion of aspect-oriented software development is best left to books dedicated to the subject.

4.3 THE UNIFIED PROCESS(Object-oriented modeling,UML)

Ivar Jacobson, Grady Booch, and James Rumbaugh discuss the Unified Process (UP) for the need of a "use case driven, architecture centric, iterative and incremental".

Chapter 4 PROCESS MODELS

In some ways the Unified Process is an attempt to draw on the best features and characteristics of traditional software process models, but characterize them in a way that implements many of the best principles of agile software development.

4.3.1 A Brief History

During the early 1990s James Rumbaugh [Rum91], Grady Booch, and Ivar Jacobson began working on a "unified method" that would combine the best features of each of their individual object-oriented analysis and design methods and adopt additional features proposed by other experts in object-oriented modeling. The result was UML—a unified modeling language that contains a robust notation for the modeling and development of object-oriented systems. By 1997, UML became a de facto industry standard for object-oriented software development.

Chapter 4 **PROCESS MODELS**

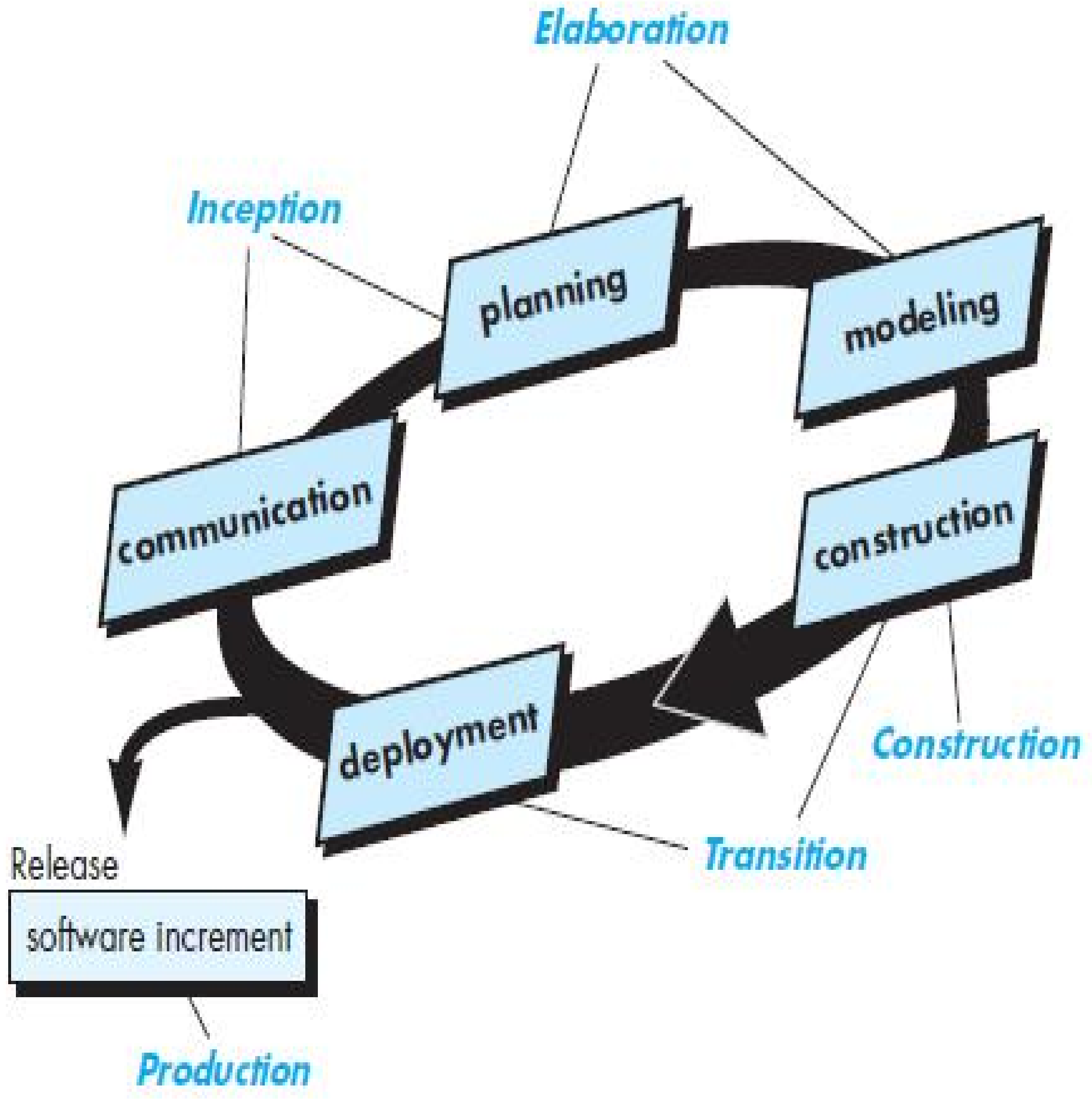
UML is used throughout Part 2 of this book to represent both requirements and design models.

4.3.2 Phases of the Unified Process

In Chapter 3, we discussed five generic framework activities and argued that they may be used to describe any software process model. The Unified Process is no exception. Figure 4.7 depicts the “phases” of the UP and relates them to the generic activities that have been discussed in Chapter 1 and earlier in this chapter.

FIGURE 4.7

The Unified Process



Chapter 4 **PROCESS MODELS**

The inception phase of the UP encompasses both customer communication and planning activities. By collaborating with stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed. Fundamental business requirements are described through a set of preliminary use cases (Chapter 8) that describe which features and functions each major class of users desires.

The elaboration phase encompasses the planning and modeling activities of the generic process model (Figure 4.7). Elaboration refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software—the use case model, the analysis model, the design

Chapter 4 PROCESS MODELS

model, the implementation model, and the deployment model. In addition, the plan is carefully reviewed at the culmination of the elaboration phase to ensure that scope, risks, and delivery dates remain reasonable.Modifications to the plan are often made at this time.

The construction phase of the UP is identical to the construction activity defined for the generic software process. Using the design model as input, the construction phase develops or acquires the software components that will make each use case operational for end users. To accomplish this, analysis and design models that were started during the elaboration phase are completed to reflect the final version of the software increment. All necessary and required features and functions for the software increment (i.e., the release) are then implemented in source code. As components are

Chapter 4 PROCESS MODELS

being implemented, unit tests are designed and executed for each. In addition, integration activities (component assembly and integration testing) are conducted. Use cases are used to derive a suite of acceptance tests that are executed prior to the initiation of the next UP phase.

The transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity. Software is given to end users for beta testing, and user feedback reports both defects and necessary changes. In addition, the software team creates the necessary support information (e.g., user manuals,troubleshooting guides, installation procedures) that is required for the release. At the conclusion of the transition phase, the software increment becomes a usable software release.

Chapter 4 PROCESS MODELS

The production phase of the UP coincides with the deployment activity of the generic process. During this phase, the ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated.

It is likely that at the same time the construction, transition, and production phases are being conducted, work may have already begun on the next software increment. This means that the five UP phases do not occur in a sequence, but rather with staggered concurrency.

4.4.1 Personal Software Process(read it by yourselves)

4.4.2 Team Software Process(read it by yourselves)

Chapter 4 PROCESS MODELS

4.5 PROCESS TECHNOLOGY

One or more of the process models discussed in the preceding sections must be adapted for use by a software team. To accomplish this, process technology tools have been developed to help software organizations analyze their current process, organize work tasks, control and monitor progress, and manage technical quality.

Process technology tools allow a software organization to build an automated model of the process framework, task sets, and umbrella activities discussed in Chapter 3. The model, normally represented as a network, can then be analyzed to determine typical workflow and examine alternative process structures that might lead to reduced development time or cost.

Chapter 4 PROCESS MODELS

Once an acceptable process has been created, other process technology tools can be used to allocate, monitor, and even control all software engineering activities, actions, and tasks defined as part of the process model. Each member of a software team can use such tools to develop a checklist of work tasks to be performed, work products to be produced, and quality assurance activities to be conducted. The process technology tool can also be used to coordinate the use of other software engineering tools that are appropriate for a particular work task.

Exercise: find some Process technology tools(Commercial tools or OpenSource tools) ,and try to install and discuss them.

4.6 PRODUCT AND PROCESS

If the process is weak, the end product will undoubtedly suffer. But an obsessive overreliance on process is also dangerous.