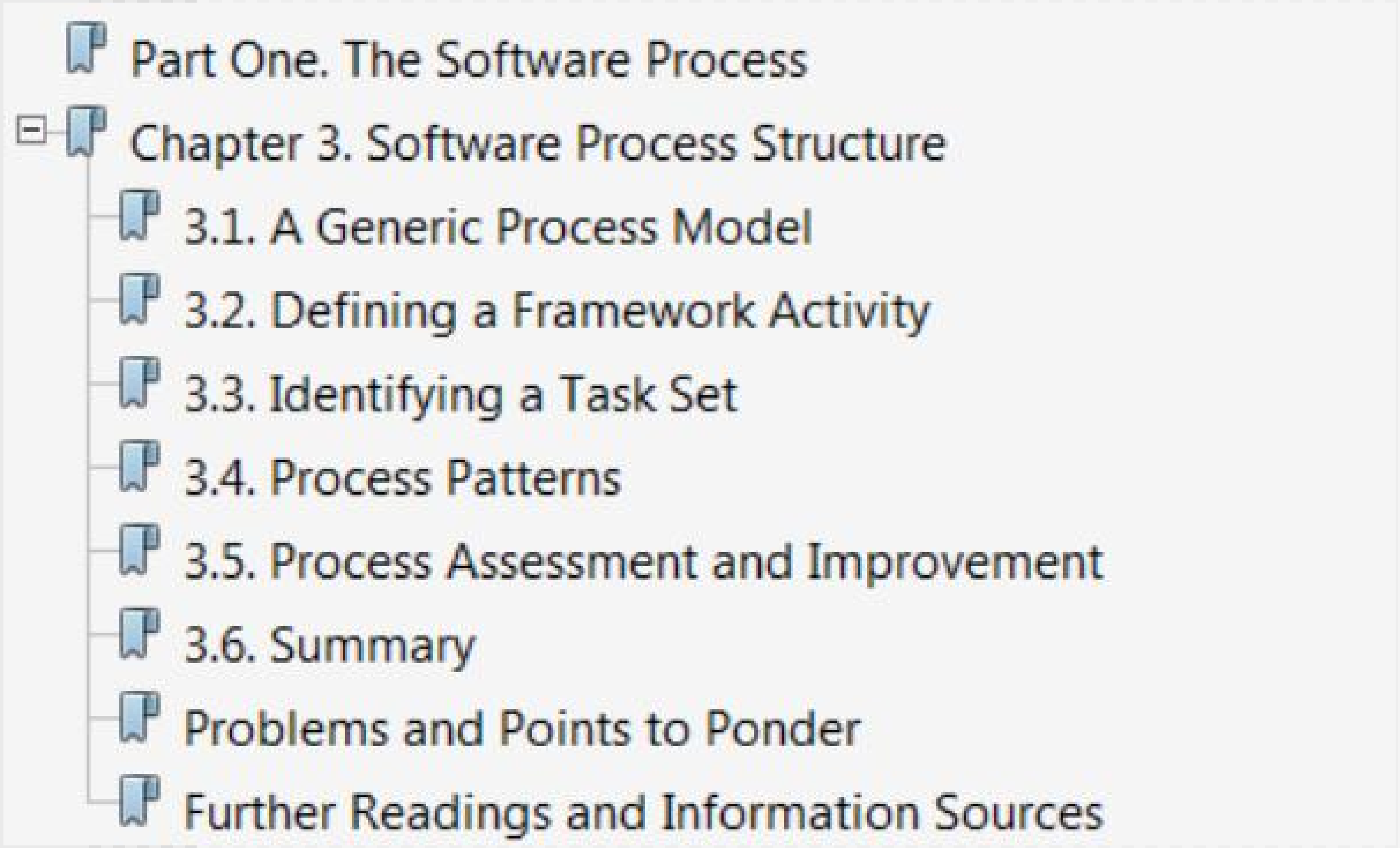
Chapter 3 THE SOFTWARE PROCESS



Chapter 3 THE SOFTWARE PROCESS

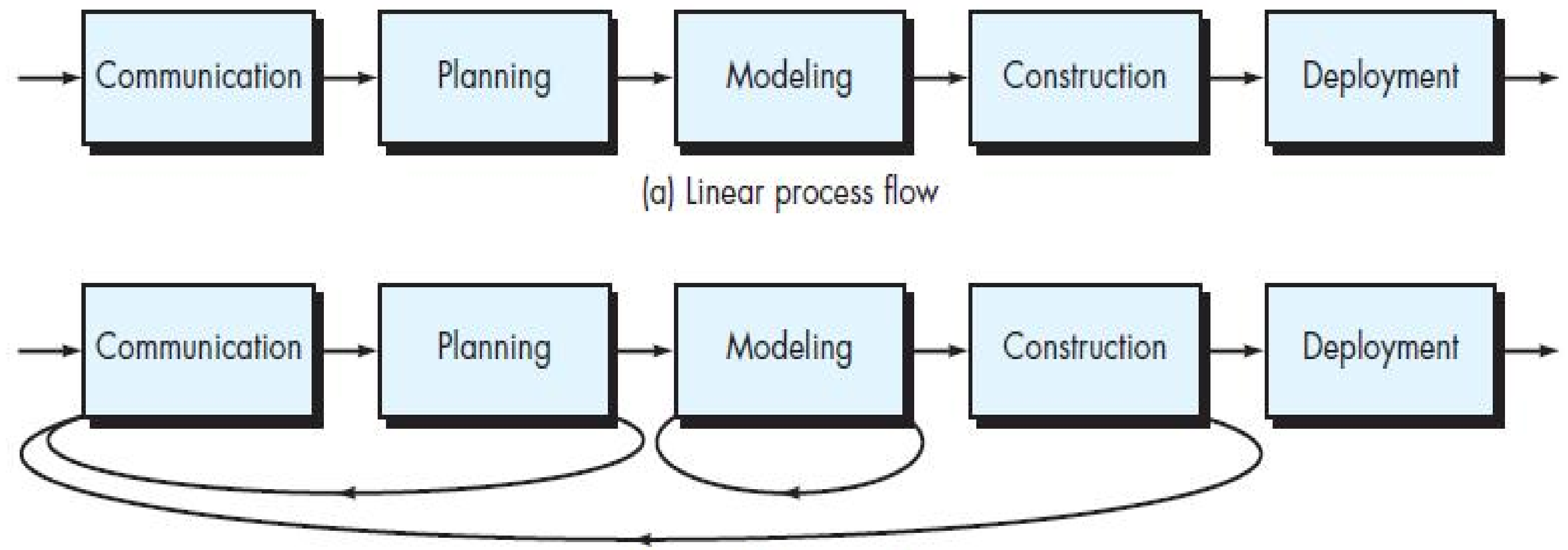
3.1 A GENERIC PROCESS MODEL

A generic process framework for software engineering defines five framework activities— communication, planning, modeling, construction, and deployment . In addition, a set of umbrella activities—project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others—are applied throughout the process.

We should note that one important aspect of the software process has not yet been discussed. This aspect—called process flow — describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time and is illustrated in Figure 3.2.

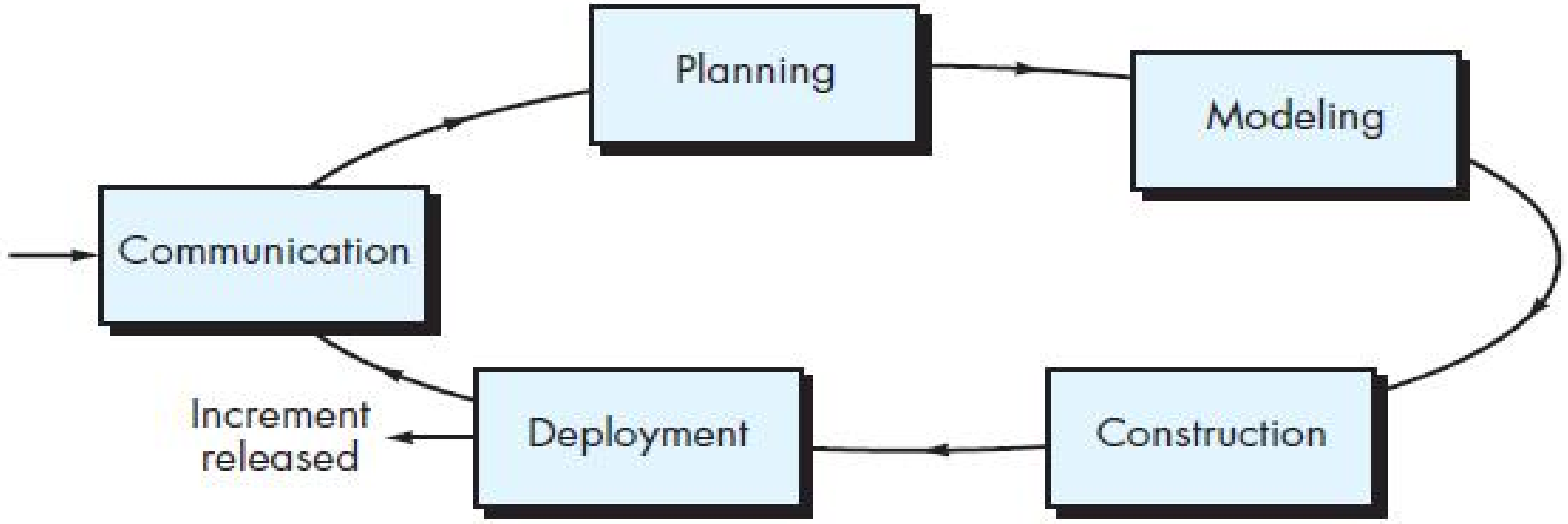
Chapter 3 THE SOFTWARE PROCESS

FIGURE 3.2 Process flow

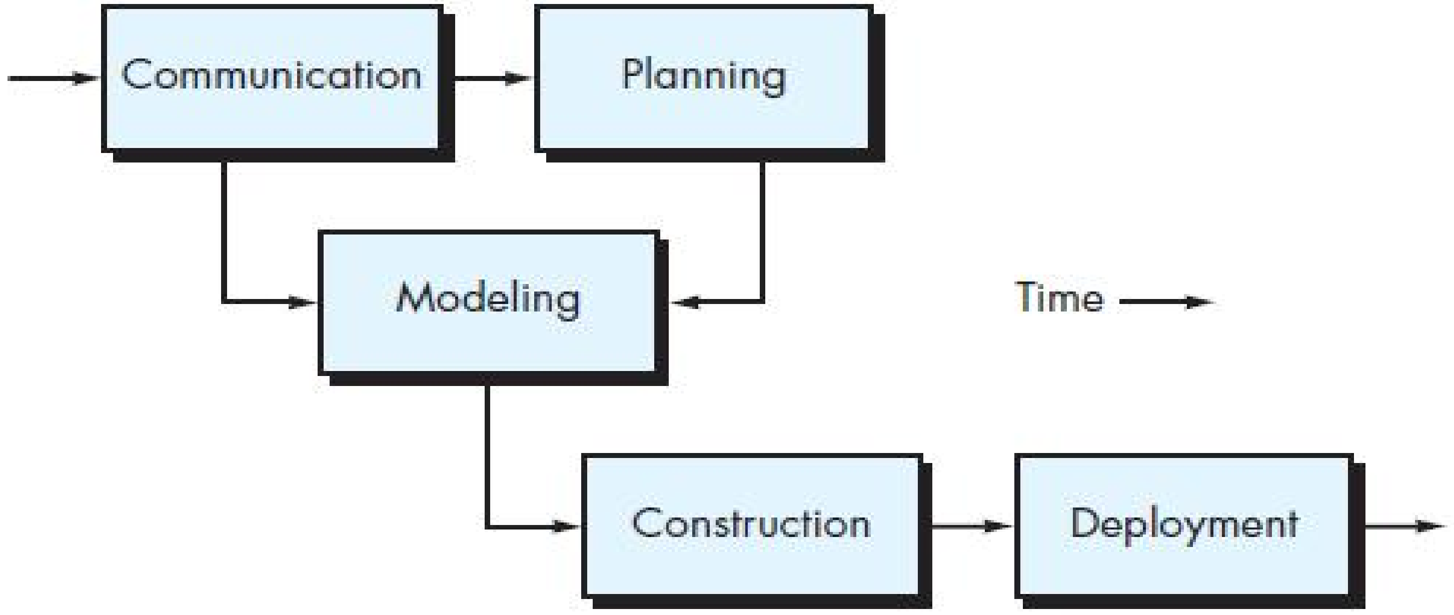


(a) Linear process flow

(b) Iterative process flow



(c) Evolutionary process flow



(d) Parallel process flow

Chapter 3 THE SOFTWARE PROCESS

3.2 DEFINING A FRAMEWORK ACTIVITY

A software team would need significantly more information before it could properly execute any one of these activities as part of the software process. Therefore, you are faced with a key question: What actions are appropriate for a framework activity, given the nature of the problem to be solved, the characteristics of the people doing the work, and the stakeholders who are sponsoring the project?

For a small software project requested by one person (at a remote location) with simple, straightforward requirements, the communication activity might encompass little more than a phone call or email with the appropriate stakeholder. Therefore, the only necessary action is phone conversation, and the work tasks (the task set) that this action encompasses are:

Chapter 3 THE SOFTWARE PROCESS

① Make contact with stakeholder via telephone.

② Discuss requirements and develop notes.

③ Organize notes into a brief written statement of requirements.

④ Email to stakeholder for review and approval.

If the project was considerably more complex with many stakeholders, each with a different set of (sometime conflicting) requirements, the communication activity might have six distinct actions (described in Chapter 8): inception, elicitation, elaboration, negotiation, specification, and validation. Each of these software engineering actions would have many work tasks and a number of distinct work products.

3.3 IDENTIFYING A TASK SET

Chapter 3 THE SOFTWARE PROCESS

Each software engineering action (e.g., elicitation, an action associated with the communication activity) can be represented by a number of different task sets —each a collection of software engineering work tasks, related work products, quality assurance points, and project milestones.

You should choose a task set that best accommodates the needs of the project and the characteristics of your team. This implies that a software engineering action can be adapted to the specific needs of the software project and the characteristics of the project team.

Exercise: discuss the following example.

INFO

Task Set

A task set defines the actual work to be done to accomplish the objectives of a software engineering action. For example, *elicitation* (more commonly called "requirements gathering") is an important software engineering action that occurs during the **communication** activity. The goal of requirements gathering is to understand what various stakeholders want from the software that is to be built.

For a small, relatively simple project, the task set for requirements gathering might look like this:

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty.

For a larger, more complex software project, a different task set would be required. It might encompass the following work tasks:

1. Make a list of stakeholders for the project.
2. Interview each stakeholder separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder input.
4. Schedule a series of facilitated application specification meetings.
5. Conduct meetings.
6. Produce informal user scenarios as part of each meeting.
7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system.

Both of these task sets achieve "requirements gathering," but they are quite different in their depth and formality. The software team chooses the task set that will allow it to achieve the goal of each action and still maintain quality and agility.

Chapter 3 THE SOFTWARE PROCESS

3.4 PROCESS PATTERNS

Every software team encounters problems as it moves through the software process. It would be useful if proven solutions to these problems were readily available to the team so that the problems could be addressed and resolved quickly.

A process pattern describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solutions to the problem.

Stated in more general terms, a process pattern provides you with a template—a consistent method for describing problem solutions within the context of the software process.

Patterns can be defined at any level of abstraction. In some cases,

Chapter 3 THE SOFTWARE PROCESS

a pattern might be used to describe a problem (and solution) associated with a complete process model (e.g., prototyping). In other situations, patterns can be used to describe a problem (and solution) associated with a framework activity (e.g.,unit testing) or an action within a framework activity (e.g., project estimating).

Ambler has proposed **a template** for describing a process pattern:

Pattern Name. The pattern is given a meaningful name describing it within the context of the software process(e.g., TechnicalReviews).

Forces. The environment in which the pattern is encountered and the issues that make the problem visible and may affect its solution.

Type. The pattern type is specified. Ambler suggests three types:

Chapter 3 THE SOFTWARE PROCESS

① **Stage pattern —defines a problem associated with a framework *activity* for the process. Since a framework activity encompasses multiple actions and work tasks, a stage pattern incorporates multiple task patterns (see the following) that are relevant to the stage (framework activity). An example of a stage pattern might be *EstablishingCommunication*. This pattern would incorporate the task pattern *RequirementsGathering* and others.**

② **Task pattern —defines a problem associated with a software engineering *action* or *work* task and relevant to successful software engineering practice (e.g., Prioritizing Requirement is a task pattern).**

③ **Phase pattern —define *the sequence of framework activities* that occurs within the process, even when the overall flow of activities is iterative in nature. An example of a phase pattern might be SpiralModel or Prototyping.**

Chapter 3 THE SOFTWARE PROCESS

Initial Context. Describes the conditions under which the pattern applies. Prior to the initiation of the pattern: (1) What organizational or team-related activities have already occurred? (2) What is the entry state for the process? (3) What software engineering information or project information already exists?

For example, the Planning pattern or requirement analysis modeling (a stage pattern) requires that (1) customers and software engineers have established a collaborative communication; (2) successful completion of a number of task patterns [specified] for the Communication pattern has occurred; and (3) the project scope, basic business requirements, and project constraints are known.

Problem. The specific problem to be solved by the pattern.

Solution. Describes how to implement the pattern successfully.

Chapter 3 THE SOFTWARE PROCESS

Resulting Context. Describes the conditions that will result once the pattern has been successfully implemented. Upon completion of the pattern: (1) What organizational or team-related activities must have occurred? (2) What is the exit state for the process? (3) What software engineering information or project information has been developed?

Related Patterns. Provide a list of all process patterns that are directly related to this one. This may be represented as a hierarchy or in some other diagrammatic form. For example, the stage pattern Communication encompasses the task patterns: *ProjectTeam, CollaborativeGuidelines, ScopeIsolation, RequirementsGathering, ConstraintDescription, and ScenarioCreation.*

Known Uses and Examples. Indicate the specific instances in

Chapter 3 THE SOFTWARE PROCESS

which the pattern is applicable. For example, Communication is mandatory at the beginning of every software project, is recommended throughout the software project.

Process patterns provide an effective mechanism for addressing problems associated with any software process. The patterns enable you to develop a hierarchical process description that begins at a high level of abstraction (a phase pattern). The description is then refined into a set of stage patterns that describe framework activities and are further refined in a hierarchical fashion into more detailed task patterns for each stage pattern. Once process patterns have been developed, they can be reused for the definition of process variants

Exercise: discuss the following example.

INFO

An Example Process Pattern

The following abbreviated process pattern describes an approach that may be applicable when stakeholders have a general idea of what must be done but are unsure of specific software requirements.

Pattern Name. **RequirementsUnclear**

Intent. This pattern describes an approach for building a model (a prototype) that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.

Type. Phase pattern.

Initial Context. The following conditions must be met prior to the initiation of this pattern: (1) stakeholders have been identified; (2) a mode of communication between stakeholders and the software team has been established; (3) the overriding software problem to be solved has been identified by stakeholders; (4) an initial understanding of project scope, basic business requirements, and project constraints has been developed.

Problem. Requirements are hazy or nonexistent, yet there is clear recognition that there is a problem to be solved, and the problem must be addressed with a software solution. Stakeholders are unsure of what they want; that is, they cannot describe software requirements in any detail.

Solution. A description of the prototyping process would be presented here and is described later in Section 4.1.3.

Resulting Context. A software prototype that identifies basic requirements (e.g., modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, (1) the prototype may evolve through a series of increments to become the production software or (2) the prototype may be discarded and the production software built using some other process pattern.

Related Patterns. The following patterns are related to this pattern: **CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.**

Known Uses and Examples. Prototyping is recommended when requirements are uncertain.

Chapter 3 THE SOFTWARE PROCESS

3.5 PROCESS ASSESSMENT AND IMPROVEMENT

The existence of a software process is no guarantee that software will be delivered on time, that it will meet the customer's needs, or that it will exhibit the technical characteristics that will lead to long-term quality characteristics. Process patterns must be coupled with solid software engineering practice. In addition, the process itself can be assessed to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering.

A number of different approaches to software process assessment and improvement have been proposed over the past few decades:

Chapter 3 THE SOFTWARE PROCESS

Standard CMMI Assessment Method for Process Improvement (SCAMPI) —provides a five-step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting, and learning. The SCAMPI method uses the SEI CMMI as the basis for assessment.

CMM-Based Appraisal for Internal Process Improvement (CBA IPI) —provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment.

SPICE (ISO/IEC15504) —a standard that defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process.

Chapter 3 THE SOFTWARE PROCESS

ISO 9001:2000 for Software— a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies.

3.6 SUMMARY

A generic process model for software engineering encompasses a set of framework and umbrella activities, actions, and work tasks；

Each of a variety of process models can be described by a different process flow—a description of how the framework activities, actions, and tasks are organized；

Process patterns can be used to solve common problems that are encountered as part of the software process.