Chapter 1
Introduction

This chapter motivates the need for defining and managing software process models. Basic concepts and terminology are presented. Figure 1.1 displays the chapter structure.

![Chapter structure diagram](image)

**Fig. 1.1** Chapter structure

### 1.1 Objectives of This Chapter

After reading this chapter, you should be able to:

- Understand the reasons why software processes are important for software development projects and organizational learning
- Name the different goals of software process modeling
- Appreciate the need for software process modeling, process management, and process improvement in organizations that are developing, operating, or maintaining software-based systems or services
- Explain the basic terms

### 1.2 Motivation

Nowadays, the business of many companies and organizations is essentially based on software. Software-intensive systems, such as automotive or telecommunication systems, and services, such as financial services, increasingly depend on software. Software adds significant value to many products and services and allows for
competitive differentiation in the market. The increasing importance of software as well as new software development paradigms such as model-driven or lean software development and future software-based applications impose many challenges and demands on software development, operation, and maintenance. In the following, several reasons are given for why organizations should place an emphasis on process management.

Typically, software and software-intensive systems are developed with hundreds or thousands of people in teams. They perform a multitude of different activities, so-called processes. Systematic coordination and cooperation mechanisms are needed in order to successfully create customer value and fulfill project goals under given project constraints such as budget limitations or deadlines. Descriptions of processes, so-called process models, are a necessary means for coordinating such endeavors. Process models can be used to define work procedures, prescribe the interfaces between tasks, support the organization of work products, or support the management of necessary resources. Team-based development has several characteristics that are challenging to deal with when conducting projects. Some typical characteristics are:

- Many activities are not performed by individuals, but are shared among different developers working together smoothly.
- In large projects, a multitude of activities can be performed in parallel. This requires good coordination so that the results of these tasks fit together in a planned way.
- There are many relationships between activities. Documents or code, for instance, can be exchanged between activities or may be used jointly by different activities. In addition, temporal dependencies may exist between activities.
- Many activities need to be synchronized so that they contribute to overall project goals. In systems engineering, for instance, software engineering processes often need to be synchronized with processes from mechanical and electrical engineering.
- As software development is a largely human-based activity, building teams is an important issue. Teamwork involves, for instance, selecting a team, harmonizing the contributions of individual members, integrating different skills and interests, and solving conflicts. Clear responsibilities can help to overcome problems with team development.
- Managing human-based processes requires great leadership skills. One of the main tasks is to motivate people to contribute to common goals.
- Besides product requirements, project managers have to consider process requirements when performing projects and leading teams. Examples of process requirements include adherence to process standards or required productivity.

Software and systems development is being increasingly performed concurrently in different countries with many customer–supplier relationships along the development chain. Outsourcing, offshoring, and nearshoring are aggravating this trend. Global software development is close to becoming the norm. Motivators for globally distributed software development are [1]:
Limited trained workforce in technologies that are required to build today’s complex systems
- Differences in development cost that favor geographical dispersal of teams
- A “shift”-based work system facilitated by time zone differences allowing for shorter times to market
- Advances in infrastructure (e.g., availability of Internet bandwidth and software development and integration tools)
- A desire to be “close” to a local market

It is inherently more difficult to coordinate projects where teams are physically distributed than projects with colocated teams. This is mainly due to the lack of implicit knowledge shared among developers who work in colocated environments. In addition, different cultures significantly aggravate the coordination of large software development projects and lead to manifold new coordination mechanisms. Supporting such distributed development requires well-understood and accurately implemented process interfaces and process synchronization. In addition, tasks need to be distributed among different sites (Fig. 1.2).

Large systems usually consist of components from different disciplines (e.g., electrical engineering, mechanical engineering, software engineering). In addition, software-based systems penetrate more and more areas of our daily life, which means that these systems must be easy to use for nonexperts. Hence, disciplines such as sociology and psychology are getting increasingly relevant for software development. In consequence, specialists from many disciplines have to work together when developing, maintaining, or operating software-based systems and services. Historically, software engineering has, to a large extent, evolved separately from other disciplines. Other disciplines have developed their own terminology, methods, techniques, tools, culture, and way of solving problems.
Therefore, integrated development between different disciplines requires a very careful understanding of the other disciplines and, as a minimum, harmonized and synchronized interfaces between the processes of the different disciplines.

More and more organizations are deploying systematic improvement programs. Often they follow so-called process capability or maturity models such as ISO/IEC 15504 [2] and CMMI [3]. The reasons for this are, for instance, that some organizations are forced to demonstrate their capabilities (such as for winning a bid) or that organizations use these maturity models as a framework for their improvement activities. At a certain level, these maturity models typically require the existence of explicit processes; on higher levels, they require capabilities for managing these processes in a quantitative way.

Nowadays, an increasing number of organizations are forced to adhere to regulatory constraints that require the presence of explicit processes and the demonstration of adherence to those processes. Examples are the IEC 61508 standard [4] for safety-related systems and the tailoring of European Cooperation for Space Standardization (ECSS) software engineering standards [5] for ground segments [6] at European Space Agency (ESA).

One of today’s most important challenges is that software is taking over more and more critical functionality. Therefore, software failures have a large potential for causing economic or even physical harm. Software is currently becoming the major source of critical system failures. This implies that the software included in many systems and services needs to be developed, operated, and maintained in such a way that critical qualities such as reliability, safety, security, privacy, or robustness can be assured at acceptable levels. Since many important critical product requirements cannot be fulfilled by features added to a system already developed, these requirements have to be considered systematically throughout the entire development process. As a consequence, activities regarding the assurance of these desired qualities need to be integrated into the overall development process, understood with respect to their effects on the resulting system, and adhered to during project performance.

Market dynamics require an organization to adapt better and faster to changes in the development environment, and to enforce innovations. Advanced process management is required to support assessing the impact of process changes and the flexible adaptation of processes.

All these challenges and demands on software development, operation, and maintenance require a significant transition from craft-based software development to more engineering-style software development. This addresses especially the following principles:

– Planning is based on experience
– Project execution is goal- and value-oriented and adheres to defined processes
– Projects are traceable and controllable
– Relevant process effects are predictable
– Learning and improvement cycles are established
These principles are widely accepted and established in traditional disciplines such as production engineering or mechanical engineering. In applying these principles to software engineering, one needs to consider the specifics of software (e.g., software is developed rather than produced; the effects of techniques depend on the development environment; software development involves many creative activities; data is less frequent and mostly of a nonparametric nature).

There are several approaches to applying engineering principles to software development that aim at so-called disciplined software development, including the problem-oriented Quality Improvement Paradigm (QIP) [7] and the solution-oriented Capability Maturity Model Integration (CMMI) [3]. According to Rombach et al. [8], essential elements of these frameworks are:

- With respect to processes: defined processes, prediction models (with respect to effort, schedule, quality), analytical and constructive quality assurance processes throughout the whole lifecycle, understanding of the context-dependent aspects of key methods and techniques
- With respect to products: adequate documentation, traceable documentation, evolvable architecture
- With respect to management: adequate workforce capabilities and staffing, sufficient continuing education and training, guaranteeing the sustainability of core competencies
- With respect to organizational improvement: traceable quality guidelines, comprehensive configuration management, learning organization

Understanding and gaining intellectual control over processes is a prerequisite for managing, controlling, and optimizing the development and evolution of software-intensive systems and services. This implies the establishment of advanced process management capabilities and an adequate understanding of the impact of processes on the generated products, services, and business values in different situations.

Due to the importance of software process models, organizations should have adequate process management capabilities in place to define, use, and evolve process models. Insufficient process management can lead to serious failures, including inefficient productivity, increased time to market, and decreased workforce motivation. If no adequate process management is established, this typically causes problems such as

- Unnecessary rework
- Deviations from plan are detected too late
- Confusion regarding roles and responsibilities
- Documents cannot be found when needed because they are not associated with process steps
- Variations in process execution
- Permanently incomplete and inconsistent process documentation
 Deferred certification because appropriate process documentation cannot be produced promptly
 Performance inefficiency (due to “unfit for purpose” processes)
 Uncertain execution and dependence on individual efforts (due to vague and incomplete process descriptions)
 Inefficient division of work and double work (due to poorly defined interfaces)

Hence, the typical question is no longer if process management is necessary, but how to define and implement a strategy for introducing advanced process management step by step and how to evaluate its success.

1.3 Software Process Modeling and Improvement

Following Osterweil [9], process models can be seen as generalized solution specifications that can be instantiated to perform concrete activities: While a process is a vehicle for solving problems and achieving development goals, a process model is a specification on how this is done. Process models can be used for different purposes, e.g., for coordinating, synchronizing, monitoring, and improving software development, maintenance, and operation activities.

There is no set of ideal process models that can be used for the development or evolution of software-intensive systems and services. The suitability of a process model heavily depends on the so-called context of a project, i.e., the characteristics of a development environment and the goals of a project. The effectiveness of a specific testing process, for instance, can highly depend on context characteristics, like the required reliability level of the test object, the experience of the test team, the budget for testing, the application domain, and other factors.

Choosing appropriate process models and tailoring them for a specific project and development environment is important and requires sufficient understanding of the effects of the processes in this very environment. This, in turn, requires an understanding of the cause-effect relationship between processes and products under typical conditions of the development environment. Therefore, development organizations should invest effort into determining the effects of processes in their own environment. Empirical studies and simulation are means to gaining such knowledge.

The need for software process improvement (SPI) is being widely recognized nowadays. Due to the fact that software development processes are usually human-based and depend on the development context, changes to these processes typically cause significant costs and should be considered carefully. Alternative improvement options need to be evaluated with respect to their implementation cost and their potential impact on business goals.

The field of software process modeling, analysis, and evolution is also an important research area. This is especially motivated by the following:
– Software engineering methods, techniques, and tools are being used in processes (i.e., the processes form the prerequisites for their successful use). Hence, research on methods, techniques, and tools requires an understanding of how they are being used. Appropriate processes are a critical success factor for gaining benefits from research results. Researchers who are not familiar with processes in which their research results are being used will likely fail to produce beneficial results.

– Processes need to be investigated in order to identify and assess strengths and weaknesses and to identify and evaluate improvements. Due to the fact that many processes are human-based activities, their behavior is nondeterministic, and the effects of processes need to be studied empirically for specific contexts.

– There are still many problems and challenges related to process management that lead to fundamental research questions (e.g., how to support the replanning of human-based processes, how to provide process models for reuse, how to define the degree of allowed flexibility).

### 1.4 Process Modeling Goals and Benefits

Software process modeling supports a wide range of objectives. Based on Curtis et al. [10], the following basic objectives for software process modeling can be observed:

– Facilitate human understanding and communication
– Support process improvement
– Support process management
– Provide automated guidance in performing process
– Provide automated execution support

Among others, the following benefits are expected from systematic process modeling:

– Better transparency of software engineering activities
– Reduced complexity of large development efforts
– The ability to perform process measurement (i.e., process models that are used in practice are a prerequisite for process measurement and, in consequence, for process improvement)
– The ability to undergo process assessments (i.e., explicitly defined process models are a prerequisite for demonstrating process maturity)
– Predictability with respect to the process characteristics and the characteristics of the results is only achievable with explicit models (i.e., enabling predictability for characteristics such as consumed effort, completion date, or reliability of a produced software component requires the existence of explicit process models, although this is not enough and other models are needed, too)
1.5 Terminology

Compared to other engineering disciplines, software engineering and especially software process modeling is quite a young discipline. Currently, a mature or standardized terminology does not exist. Besides the newness of the domain, this is mainly caused by the parallel emergence of different process notations and influences from different other domains such as business process modeling or programming notations.

In practice, organizations often use different terms for similar constructs (e.g., activity, task, work assignment, work package, step, ...), or people mix terms that describe models with terms that describe real objects (e.g., using the term “software process” instead of “software process model”). Often the domain is unspecified (e.g., using the term “process” instead of “software process”). However, a common understanding of terms does exist. In the case of imprecise usage of terms, the semantics can often be determined by the context.

In the following, we present a terminology that can be considered as a kind of common sense in the process modeling community. This terminology is mainly based on [10–12], and to a minor degree on [9, 13] and [14], as well as on other publicly available sources.

A software process is a goal-oriented activity in the context of engineering-style software development.

Examples are the creation of a product (e.g., coding of system component no. 15 in project Alpha at company ACSoft), testing of a system, measurement of a code module, planning of a project, or packaging of experience for reuse in future projects.

Typical characteristics of software processes are:

- They are enacted in the real world
- They usually transform one or more input products into one or more output products by consuming further products (e.g., guidelines)
- They can be performed by humans (“enactment“) or machines (“execution“) or both together
- They can be refined by subprocesses, each of which can also be refined

Often, software processes are also called “software development processes”. We recommend using the term software process because (a) many processes are not directly related to development (such as maintenance processes), and (b) software processes are also relevant in areas where not only software is produced (e.g., when building software-based systems).

In this book, if the context is clear, the term “process” is used instead of “software process.”
A project is a unique endeavor, which is limited by a start date and an end date and should achieve a goal.

It should be recognized that permanent or routine activities are not projects.

A project phase (short: phase) is a collection of logically separated project activities, usually culminating in the completion of a major deliverable or the achievement of a major milestone.

Typical characteristics of project phases are:
- Phases are mainly completed sequentially, but can overlap in some project situations
- Phases can be subdivided into subphases
- Unlike a process, a phase is always defined by a start date and an end date. If this period is finished, the phase is finished. Typically, processes can be activated multiple times
- Typical examples of phases are the elaboration phase, the construction phase, or the transition phase. Phases are usually used when looking at a project from a management perspective

A major reason why it is important to differentiate between a process and a project phase is that there are two essentially different views on projects: a management view and an engineering view. Management often needs a period-based view on activities because investors, investments, dividends, revenue calculations, and financial plans are typically period based. Engineers usually need a product-based view on activities, i.e., a view on activities that describe in a goal-oriented way the steps needed to create, maintain, or operate a product or service.

It should be recognized that a project phase can be only performed once. If a requirements definition phase has been declared completed, it cannot be enacted again, even if there are still requirements engineering activities to be done. A process, however, can be reenacted. If, for instance, requirements defects were detected during a design review, a requirements engineering process can be reenacted in order to remove the defects (Fig. 1.3).

A model is an abstract and simplifying representation of an object or phenomenon of the real world.
Typical characteristics of models are:

- They describe only those aspects of the object or phenomenon that are (believed to be) relevant for the understanding and intended usage of the model
- They encapsulate experience and allow for an explicit representation of experience
- They can be created for different purposes such as planning, control, or prediction
- They have their own lifecycle, i.e., they can be specified, built, implemented, analyzed, used, assessed, evolved, or rejected

The frequently quoted phrase “Essentially, all models are wrong, but some are useful,” attributed to the statistician George Edward Pelham Box, highlights that a model only represents a limited number of real-world aspects and details. The challenge lies in capturing sufficient real-world aspects and details in a model so that the model can be used for its purpose.

A software process model (short: process model) is a model of a software process.

A software process model is a description of a software process. Process models are often used as a means for problem solving. The specification of the enactment of a software process by a process model is comparable to the specification of baking a cake using a recipe. Process models can be represented by using different notations (e.g., graphical, natural language, machine-readable notations).

A process model can describe a process on different levels of abstraction (e.g., lifecycle process level, engineering process level, atomic step level).
The main elements of process models are:

- A description of an identifiable activity or a group of activities
- A description of the product flow (i.e., input and output products for activities)
- A description of the control flow between processes (i.e., the enactment or execution sequence)
- A description of a refinement (i.e., the definition of a hierarchy of processes)
- A description of the relationships to techniques, methods, tools
- A description of the relationship to roles

Other process-related definitions or statements are:

**Process**: A set of partially ordered steps intended to reach a goal [12].

**Process description**: While a process is a vehicle for doing a job, a process description is a specification of how the job is to be done. Thus, cookbook recipes are process descriptions, while preparing a recipe is a process [9].

**Process model**: A software process model reflects an organization’s know-how regarding software development. Software engineering know-how has to be developed and maintained. Practical experience has shown the need for modeling software engineering entities (especially processes), measuring those entities, reusing the models, and improving the models [8].

Often, only selected elements are shown in graphical representations of process models. Figure 1.4, for instance, shows only activities, artifacts, and the product flow, while Fig. 1.5 also shows the control flow and the relationships to roles.

In the following, further basic terms are defined:

An *atomic process* (synonym: process step) is a process that does not allow further structuring in the form of subprocesses.

*Process enactment* is the performance of process steps undertaken to reach a given goal. The process performer (i.e., “agent”) can be a human or a machine. In case of a machine, the term “process execution” is usually used.

A *process definition* is a description of a process that is enactable.

Process scripts and process programs are specializations of process definitions:

A *process script* is a description of a process that is suitable for interpretation by humans. A process script should be tailored to the needs of the process performer.

(continued)
A *process program* is a description of a process that can be interpreted by machines.

A *process schema* (synonym: process metamodel, process architecture) is a conceptual framework for the consistent description of process models and their relationships. A process schema describes, on the one hand, building blocks and their relationships that form a process model, and, on the other hand, constraints on their composition.

![Fig. 1.4 Process model with product flow](image-url)
Fig. 1.5 Process model with product and control flow and roles
Until now, a single commonly accepted process schema for software development processes has not been established. Only few process management tools are flexible enough to cope with multiple process schemata or are able to import individual process schemata. Often, a process schema is created ad hoc together with the process model. This often implies description failures (e.g., phases are refined by process models).

A process agent (synonym: process performer) is a person or machine that enacts/executes the process in order to reach the process goal(s). Humans interpret process scripts, machines interpret process programs.

A process owner is a human or organizational entity that sets the goals of a process and is responsible for their achievement.

A process owner provides resources for the enactment or execution of the process and is responsible for providing appropriate process definitions.

A process engineer is a person who pursues one or several goals of process modeling (e.g., defining, extending, maintaining, improving process models).

To that end, a process engineer uses process models, which he defines, extends, improves, and manages. The process engineer should pay attention to the accuracy of the model, i.e., the correspondence between the real-world process enactment/execution and the process model.

A principle is a policy or mode of action that describes important characteristics of a process model.

Often, new process models evolve from principles. Examples of principles are:

- Active user involvement is imperative
- Frequent inspection
- Work in progress should be limited
- Timeboxed iterations
- Develop small incremental releases and iterate
- Frequent delivery of product
- Continuous integration
– Colocation
– Common coding guidelines
– Self-organizing teams
– Daily meeting

A principle is not a process or a process model, but a process or a process model can capture one or more principles. Principles should be adapted to contexts (e.g., by using experimentation) and integrated into process models. Sometimes the term “practice” is used as a synonym for principle, e.g., the principle “continuous integration” is often referred to as an XP practice.

A product is each artifact that is consumed or produced in the context of engineering-style software development.

Products can be refined by other products. Examples of products are:
– Source code
– Specification document
– Problem description
– Configuration data
– Component design
– Test case
– Test result
– Project plan

A product model is a description of a product or a class of products.

Usually, software product models consist of a description of the information units of a software product (e.g., functional requirements, nonfunctional requirements, design decisions) and a structure for arranging the information units (e.g., a table of contents for a requirements document).

The product flow consists of the relationships between products and processes that describe the access mode to the products.

The following access modes are typically defined:
– Produce (write)
– Consume (read)
– Modify (read/write)
A role is a set of processes belonging together that are assigned to one or several agents. A role combines the functional responsibility for the enactment of a process.

Examples of technical development roles are: requirements engineer, designer (architecture), designer (data/algorithms), programmer, inspector, system integration engineer, and tester.

Examples of organizational and management-oriented roles are project planner, project manager, product manager, and quality engineer.

Like a process, a role is an abstraction. A role groups competencies (i.e., knowledge and rights). Several different types of relationships between roles and agents can be defined, especially 1:1 (e.g., the tester is Mr. Miller), m:1 (e.g., Mr. Miller plays both the role of the requirements engineer and the role of the code inspector), 1:n (the role of the project manager is shared by Mr. Miller and Ms. Scott), m:n (a mixture of the previous cases). It is important that the relationship between a role and an agent is explicitly defined. This is typically done during project planning, resource planning, and replanning.

A role is typically described as a specific view on the process by:

- The activities the role is involved in (described independently of the person)
- The documents that are consumed or produced in these activities
- The level of involvement of a role in an activity
- The requirements for playing a role (e.g., qualifications)

Figure 1.6 shows an example of the activities and the product flows that relate to the role module developer.
Benefits of the role concept are:

- Activities and responsibilities of persons involved in a project can be clearly defined
- Transparency supports communication and cooperation between roles
- Necessary competencies are clearly defined
- Project planning is simplified
- Resources for a project or an organization can be specified independently of available personnel
- Role bottlenecks can already be determined at an early stage during project planning

The relationship between roles and process models can be qualified: For instance, a role can perform a process or a role needs to be informed when a process is performed or completed. Often, a so-called “Responsibility Assignment Matrix” (RAM) is used to qualify the relationship between roles and processes.

Finally, we define the term project plan, which integrates and instantiates several of the concepts presented above:

A project plan is a specification of the necessary resources for the execution of a process definition, the relationships between these resources and processes, the produced products including the product flows, and restrictions of any type concerning the execution of the process.

References